Secondo Progetto Laboratorio di Programmazione
Gestione dinamica della memoria

Stefano Guerrini

A.A. 2001/02

L'obiettivo del progetto è implementare le funzioni per l'allocazione/deallocazione dinamica della memoria. In pratica, si dovrà implementare un modulo che esporta due funzioni `mymalloc` e `myfree` equivalenti alle `malloc` e `free` di sistema - la sola differenza sarà dove e come queste funzioni prendono la memoria da allocare.

In linea di principio, dopo aver creato il proprio modulo `mymem.c` con header `mymem.h`, dovrebbe essere possibile rimpiazzare tutte le chiamate di `malloc` e `free` con chiamate a `mymalloc` e `myfree` in ogni programma che alloca dinamicamente la memoria solo mediante la `malloc`, (ovviamente, dopo aver aggiunto la include di `mymem.h` e supponendo di compilare e collegare anche il modulo `mymem.c` al momento della compilazione).

La memoria su cui le funzioni `mymalloc` e `myfree` operano è un vettore `mymem` di `DIM_MYMEM` byte. Per ogni richiesta di memoria, la `mymalloc` alloca un blocco di memoria di dimensione almeno pari a quella richiesta dall'utente e ritorna il puntatore alla prima locazione utile del blocco (lo header del blocco non deve essere visto dall'utente).

La `myfree` dealloca un blocco di memoria precedentemente allocato da una `mymalloc` rendendolo nuovamente disponibile per successive allocazioni. Un blocco deallocato diviene un blocco libero. Tutti i blocchi liberi della memoria vengono mantenuti nella cosiddetta *lista libera*. Per questo, è necessario che nello header di ogni blocco libero, oltre alla dimensione del blocco, sia mantenuto il puntatore al successivo blocco nella lista libera.

A questo punto possiamo essere più precisi sul compito di `mymalloc` e `myfree`- almeno nel'approccio più semplice al problema.

`mymalloc`: Cerca all'interno della lista libera un blocco libero di dimensione maggiore o uguale a quella del blocco da allocare. (La politica da seguire nella scelta di tale blocco è lasciata libera. In particolare, si consiglia di scegliere tra *first fit*, si prende il primo blocco della lista libera sufficientemente grande, e *best fit*, si prende il più piccolo blocco libero nel quale sia possibile allocare il blocco richiesto.) Dopo aver trovato un blocco libero sufficientemente grande, lascia nella lista libera la parte del blocco che rimane inutilizzata e ritorna il puntatore alla prima locazione utile del blocco allocato.

`myfree`: Inserisce il blocco deallocato nella lista libera.

Frammentazione e compattazione della memoria libera

La situazione iniziale della memoria è un unico blocco libero di dimensione pari a `DIM_MYMEM`. Dopo un certo numero di operazioni sulla memoria, la memoria libera risulta formata da un insieme in blocchi liberi. Alcuni di tali blocchi liberi potrebbero essere contigui, di conseguenza, potrebbero essere fusi per dare origine ad un blocco libero di dimensione pari alla somma delle loro dimensioni. Questa *frammentazione* della memoria libera può portare a situazioni in cui una semplice scansione della lista libera porterebbe a concludere erroneamente l'impossibilità di allocare un blocco richiesto dall'utente, visto che la dimensione di tale blocco è maggiore di ogni blocco nella lista libera. Invece, il blocco richiesto dall'utente potrebbe essere allocato dopo una *compattazione* della memoria libera che porti a compattare in un unico blocco ogni sequenza di blocchi liberi contigui. Una semplice configurazione in cui si ha una tale situazione è quella derivante dall'allocazione di un blocco di dimensione `DIM_MYMEM`/2 seguita dalla sua deallocazione; la memoria, pur essendo tutta libera, risulta suddivisa in due blocchi di dimensione `DIM_MYMEM`/2, rendendo impossibile l'allocazione di un blocco di dimensione compresa tra `DIM_MYMEM`/2+1 e `DIM_MYMEM`.

Il modo più semplice per affrontare questo problema è procedere ad una compattazione della memoria libera ogni volta che ci si accorge che altrimenti non si sarebbe in grado di allocare il blocco richiesto dall'utente - ovviamente, se dopo la compattazione il blocco rimane non allocabile, la `mymalloc` dovrà rinunciare ad allocare il blocco e ritornare `NULL`. Si osservi che la compattazione dei blocchi liberi non risolve completamente il problema della frammentazione della memoria, dato che rimangono sempre possibili casi in cui, pur essendo il totale della memoria libera maggiore della dimensione del blocco da allocare, la memoria libera è suddivisa in blocchi non contigui di diimensione inferiore a quella richiesta. Purtroppo, in questi casi occorrerà rinunciare all'allocazione del blocco richiesto, dato che l'unica possibilità sarebbe spostare i blocchi già allocati per crearne uno libero di dimensione opportuna ma, in un linguaggio come il C, ciò è ovviamente impraticabile, dato che farebbbe saltare i valori dei puntatori ritornati dalle predenti chiamate di `mymalloc`.

Per risolvere il problema della framentazione dei blocchi liberi, gli studenti possono anche provare approcci diversi da quello proposto. Ad esempio, si può cercare di mantenere la lista libera priva di blocchi liberi contigui (in pratica, la compattazione viene eseguita al momento dell'inserimento dei blocchi nella lista libera), oppure, si può cercare di sfruttare la scansione della lista libera eseguita dalla `mymalloc` alla ricerca di un blocco libero sufficientemente grande per eseguire la compattazione della parte di lista scandita. In ogni caso, gli studenti dovranno provare ad analizzare vantaggi ed inconvenienti della soluzione scelta.

Cosa si deve realizzare

Un modulo `mymem.c` con header `mymem.h` che esporta le due funzioni `mymalloc` e `myfree`. Tale modulo dovrà essere scritto in modo che tutti i dettagli interni al modulo (il vettore della memoria, le funzioni e le altre variabili ausiliare) non siano visibili al suo esterno.

I file `mymem.c` e `mymem.h` dovranno essere inviati via posta elettronica a `lp13@dsi.uniroma1.it` entro il **30 giugno**.