Buddy Allocator using Bitmap

Progetto per il corso di Sistemi Operativi tenuto dal prof. Grisetti Giorgio e dalla tutor Bazzana Barbara dell'anno accademico 2020/2021.

What?

Implementazione di un Buddy Allocator utilizzando una Bitmap.

How?

Il funzionamento è quello di un tipico Buddy Allocator: la memoria viene divisa ricorsivamente in 2 (creando di fatto un albero binario) finché non si raggiunge un livello tale per cui la dimensione del *buddy* è la più piccola possibile per soddisfare la richiesta di allocazione. Dunque, se, ad esempio, avessimo 512 KB di memoria disponibile e ricevessimo una richiesta di 64 KB, la memoria verrebbe divisa una prima volta ottenendo due partizioni da 256 KB, un'ulteriore volta ottenendo 4 partizioni da 128 KB, e un'altra volta per ottenere 8 partizioni da 64 KB. A questo punto, poiché un'ulteriore divisione porterebbe ad avere 16 partizioni da 32 KB l'una (memoria più piccola di quella richiesta), ci si fermerà al livello 3 e si allocherà un blocco i della dimensione di 64KB, rendendo di fatto occupati tutti gli antenati e tutti i discendenti di i. L'albero binario ottenuto dalla divisione ricorsiva della memoria viene implementato grazie ad una *Bitmap*, ovvero una mappa (un array) di bit, in cui ogni nodo avrà valore 0 se il blocco relativo a quel nodo è libero, 1 se invece è stato allocato. In questa implementazione è stata utilizzata la convenzione secondo la quale il primo indice della bitmap (ovvero la radice dell'albero) vale 0. Poiché, come detto in precedenza, il risultato della divisione ricorsiva non è nient'altro che un albero binario, ciò semplifica di molto le operazioni fatte a livello di programmazione. Sarà infatti semplicissimo riuscire ad accedere all'indice di un *buddy* dato un nodo i (se ha valore 0 non avrà *buddy* (sarà la root); se i è pari, allora i-1 sarà l'indice del suo *buddy*, mentre se i è dispari, il suo *buddy* avrà indice i+1), così come sarà semplice riuscire a trovare il primo e l'ultimo indice di un livello dato un livello L (il primo indice sarà (2^{L})-1, mentre l'ultimo sarà $(2^{(L+1)})$-2). Inoltre è possibile riuscire a risalire all'indice del padre dato un nodo i (basterà prendere la parte intera dell'operazione (i-1)/2), mentre per risalire al livello in cui si trova l'indice i basterà prendere la parte intera del $\log_2(i+1)$. Il programma fa uso di 2 struct, una struct **BuddyAllocator** che conserva informazioni quali la BitMap relativa al Buddy Allocator, il numero di livelli, la dimensione minima di memoria che si può richiedere, un

puntatore ad un buffer che rappresenta l'area di memoria da gestire e la sua relativa dimensione, e una struct **BitMap**, che contiene un buffer, la dimensione del buffer e il numero di bit della bitmap. Quando viene richiesto un blocco di memoria, si calcola innanzitutto il livello in cui le partizioni hanno dimensione sufficiente ad allocare tale blocco di memoria. In seguito, se al livello corrispondente viene trovato un *buddy* libero (ipotizziamo di indice j), allora verrà allocata la memoria, verrà settato a 1 il bit relativo al blocco j e verranno settati a 1 tutti i suoi antenati e i suoi discendenti (fino all'ultimo livello), altrimenti, se non ci sono *buddy* liberi, verrà stampato a terminale un errore MEMORY FAULT, e la memoria non sarà allocata. Inoltre, per semplificarci il lavoro quando dovremo eseguire la free del blocco, viene restituito insieme all'indirizzo in cui è stato allocato il blocco anche il relativo indice della bitmap: in questo modo si avrà che gli ultimi 4 bytes del puntatore restituitoci si riferiranno all'indice del blocco, mentre il resto si riferirà all'indirizzo della memoria allocata. Quando, quindi, effettueremo la *free* di tale blocco tramite la funzione `BuddyAllocator_free`, verrà ricavato l'indice del blocco dal puntatore restituitoci dalla funzione `BuddyAllocator_malloc`, verrà settato a 0 il bit relativo al blocco da liberare (così come quelli di tutti i suoi figli) e, se il suo *buddy* sarà libero, verrà effettuata un operazione di *merge* fino al livello più alto possibile.

How-to-run

```
$ git clone https://github.com/francesco-fortunato/ProgettoSO.git
$ cd /ProgettoSO
$ make
$ ./buddy_allocator_test x
```

dove **x** è un numero da 0 a 3:

0. Per eseguire tutti i test
1. Per eseguire un test in cui viene allocata tutta la memoria disponibile
2. Per eseguire un test in cui vengono allocati blocchi sparsi
3. Per eseguire un test in cui vengono gestiti gli errori (richiesta di memoria maggiore di quella disponibile, richiesta di deallocazione di un blocco di memoria non allocato, richiesta di allocazione di 0 bytes)

Modificare il file sorgente buddy_allocator_test.c per aggiungere, rimuovere o modificare i test, e ricompilare con l'istruzione make prima di eseguire nuovamente l'istruzione `./buddy_allocator_test x`.

Francesco Fortunato

Matricola 1848527