Progetto Sistemi Operativi: Buddy Allocator Con Bitmap

Carmella Sta Ana Pana, matricola: 1794019

What

Il progetto svolto consiste nell'implementazione di un buddy allocator utilizzando una bitmap come struttura ausiliaria di bookkeeping. Ogni bit corrisponderà a un nodo della struttura ad albero che rappresenta il buddy allocator, quindi avrò un numero di bit uguale al massimo numero di nodi/ buddies. Ogni livello corrisponde a una dimensione, cioé i blocchi che si trovano sullo stesso livello hanno tutti la stessa dimensione.
La bitmap è rappresentata nel seguente modo:

- un bit in posizione x è 0 se il blocco di memoria corrispondente è libero, ovvero il nodo di indice x è libero o almeno uno dei suoi figli è libero.
- un bit in posizione x è 1 se il blocco di memoria è occupato, e quindi il nodo di indice x è occupato o entrambi i suoi figli sono occupati.

How

Dopo aver inizializzato il buddy e la bitmap, e quindi superato tutti i controlli necessari, si possono utilizzare le due funzioni principali per l'allocazione e la deallocazione della memoria.

- **`malloc`**

La `malloc` prende come argomenti il buddy allocator e la dimensione di memoria richiesta da allocare. A tale dimensione vengono aggiunti altri 4 byte, poiché in essi verrà memorizzata l'indice corrispondente al buddy che verrà allocato.
Purché siamo ancora entro i limiti di memoria, viene calcolato il livello giusto per l'allocazione, in base proprio alla dimensione richiesta. Se non ha superato il massimo livello, può finalmente scansionare il livello dato e cercare un blocco libero. Questo sarà il compito della funzione `getBuddy`.
La `getBuddy` comincerà a controllare ogni blocco del livello partendo dal primo indice fino all'ultimo. Se l'indice corrente è occupato (quindi la bitmap ha 1 in quella posizione), passa direttamente al prossimo indice; altrimenti, il blocco è un candidato per l'allocazione. Prima però controlla che non abbia già dei figli e genitori allocati; se non è il caso, alloca settando il bit a 1 e se il fratello di tale buddy ha anche lui il bit a 1, allora setta anche il loro genitore a 1. Dopodiché, ritorna l'indice del blocco corrente alla `malloc`, il quale si occuperà di ritornare l'indirizzo di memoria corretto, aggiungendo appunto spazio per l'indice.
Se invece c'è un figlio occupato, e quindi un blocco di dimensione inferiore è stato già allocato, scansiona l'indice successivo; se ha uno o più genitori occupati, sale per vedere se trova un genitore libero e appena ne trova uno, e quindi ha un blocco libero, può spostarsi a scansionare l'indice che appartiene a quel blocco.

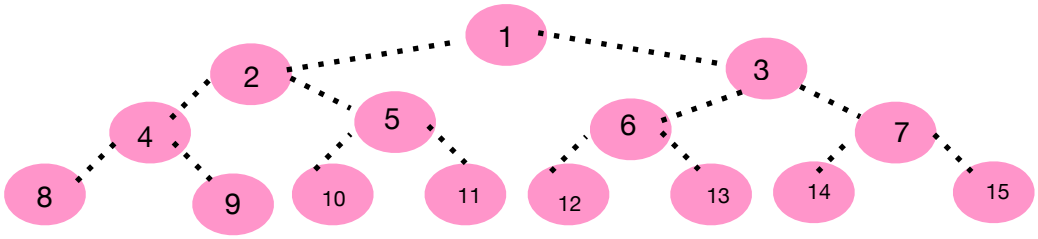
- **`free`**

La `free` controllerà se l'indirizzo di memoria passato come argomento è valido .
Poi, ricaverà l'indice del blocco da liberare, sottraendo 4 byte all'indirizzo corrente e vedendone il contenuto. Successivamente passerà alla funzione ausiliaria `releaseBuddy` l'allocator e l'indice trovato.
La `releaseBuddy` verificherà che non si stia liberando un blocco già libero o che i figli di tale blocco non siano occupati. Dopodiché setta il bit di questo indice nella bitmap a 0, e setterà a 0 anche i genitori dei livelli superiori che hanno bit 1.
Fatto ciò, il blocco si considera liberato con successo.

How to run

```
make
./allocator_test_*  (sostituire * con 1 | 2 | 3 )
./buddy_allocator_test  (test originale)
```

Esempio esecuzione: *(./allocator_test_3)*



Block sizes on each level:

512 bytes

256 bytes

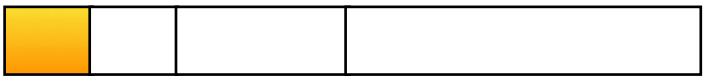
128 bytes

64 bytes

a = malloc(50) —> index 8

```
          0
      0       0
   0     0   0     0
1  0  0  0  0  0  0  0
```

Memory state:



b = malloc(200) —> index 3

```
          0
      0       1
   0     0   0     0
1  0  0  0  0  0  0  0
```

c = malloc(32) —> index 9

```
          0
      0       1
   1     0   0     0
1  1  0  0  0  0  0  0
```

d = malloc(88) —> index 5

```
          1
      1       1
   1     1   0     0
1  1  0  0  0  0  0  0
```

free(b)

```
          0
      1       0
   1     1   0     0
1  1  0  0  0  0  0  0
```

free(d)

```
          0
      0       0
   1     0   0     0
1  1  0  0  0  0  0  0
```