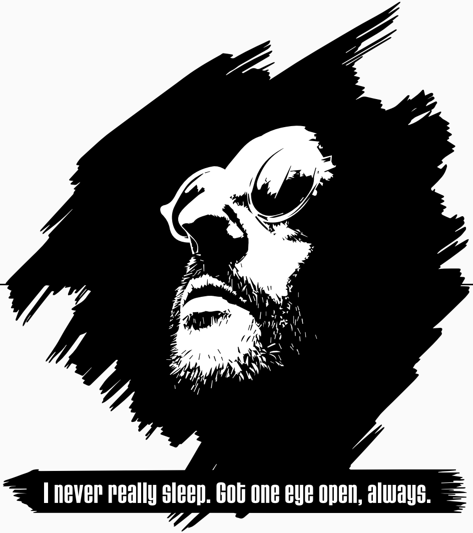
Malloc - Presentation

ACU 2020 Team



Copyright

Introduction

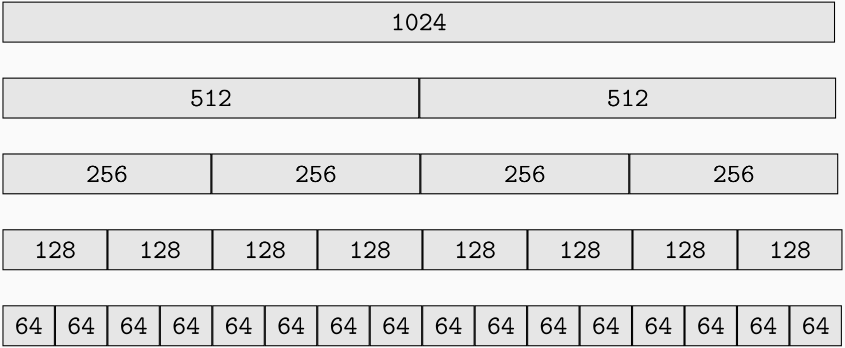
What’s different from other algorithms ?

- Low fragmentation.
- Mitigated performances on its own.
- Highly depends on the complementary strategies used with this algorithm.
- Really challenging approach.

Principle

- The idea is to manage a contiguous memory block of size n with n a power of two.
- The block can be given to the user as is or divided into two sub-blocks.
- The two sub-blocks are called buddies (each one is the buddy of the other).
- The logic is recursive and each sub-block can be given to the user or divided as well into two 2 sub-block, etc...
- Each block's size is therefore a power of two.
- You have to define a minimum size for the blocks.

Layouts



1024

512 512

256 256 256 256

128 128 128 128 128 128 128 128

64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64

Management

Allocation

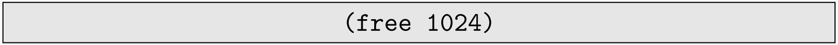
- User does a malloc of size s.
- Compute the first power of two greater than or equal to s: s2p.
- Of course s2p is greater or equal to the chosen minimum size.
- From there, find the smallest block possible of size at least s2p in the tree.
- If the free block found is at least 2 times greater than s2p, divide it into two sub-block (the division is done recursively) until the sub-block size is as small as possible and still fits s2p.

Free

- Mark the block as free.
- If the buddy of the block is also available, merge them to create a larger block.
- The merge is done recursively until it is not possible anymore.

Step 1

Start with an initial block of size 1024 bytes.

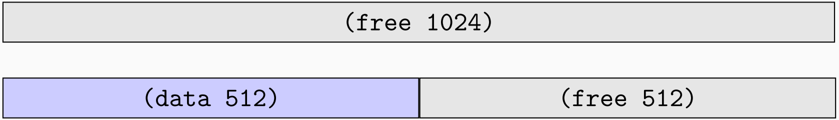


(free 1024)

Step 2

Allocation of 512 bytes:

- Split the block in two blocks of 512 bytes and use the first one.



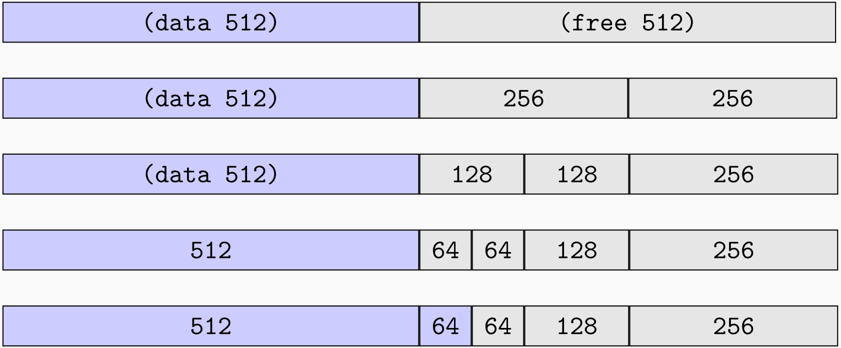
(free 1024)

(data 512) (free 512)

Step 3

Allocation of 64 bytes:

- Split the free block of 512 bytes into two blocks of 256 bytes.
- Split the first block of 256 bytes into two blocks of 128 bytes.
- Split the first block of 128 into two blocks of 64 bytes and use the first one.



(data 512) (free 512)

(data 512) 256 256

(data 512) 128 128 256

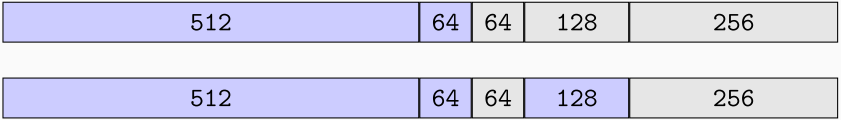
512 64 64 128 256

512 64 64 128 256

Step 4

Allocation of 120 bytes:

- The rounding power of two is 128 bytes
- Use the already available block of 128 bytes.



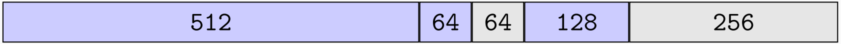
512 | 64 | 64 | 128 | 256

512 | 64 | 64 | 128 | 256

Step 5

Allocation of 500 bytes:

- The rounding power of two is 512 bytes.
- There is no block large enough: allocation is not possible.
- Would have to create a new binary buddy (not in this example).

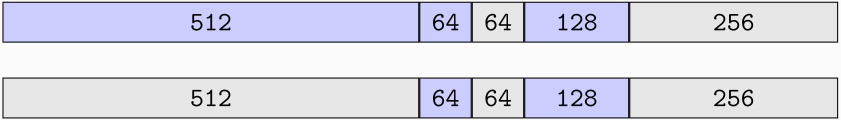


512 | 64 | 64 | 128 | 256

Step 6

Free the 512 bytes' block:

- Mark the block as free.
- The buddy of the block is not free so no merge is needed.



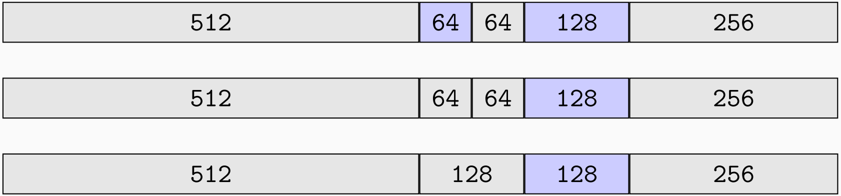
512 | 64 | 64 | 128 | 256

512 | 64 | 64 | 128 | 256

Step 7

Free the 64 bytes' block:

- Mark the block as free.
- It's buddy is also free: merge them into a free block of 128 bytes.
- The buddy of the new 128 bytes block is not free, the merge is not possible.



512 | 64 | 64 | 128 | 256

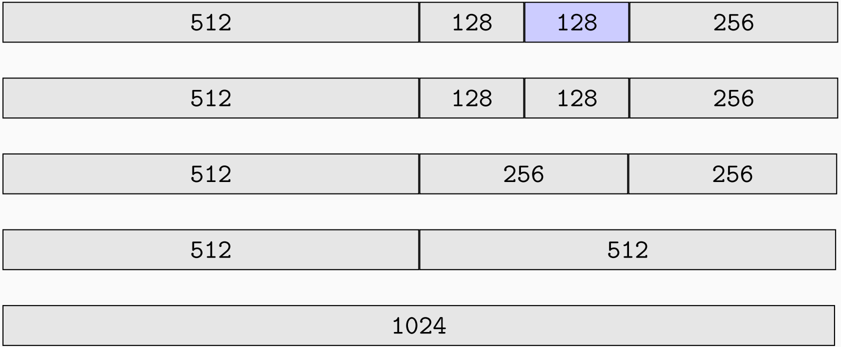
512 | 64 | 64 | 128 | 256

512 | 128 | 128 | 256

Step 8

Free 128 bytes' block:

- Mark the block as free.
- Its buddy is free: merge them into a block of 256 bytes.
- The buddy of the new 256 bytes' block is free: merge them into a block of 512 bytes.
- The buddy of the new 512 bytes' block is free: merge them into a block of 1024 bytes.



512 | 128 | 128 | 256

512 | 128 | 128 | 256

512 | 256 | 256

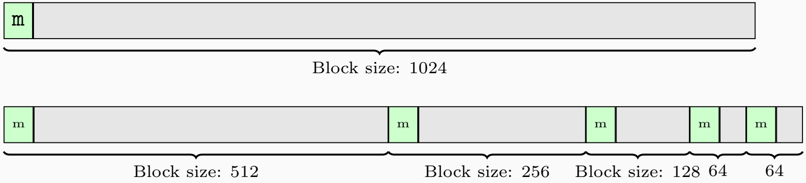
512 | 512

1024

Metadata

At the beginning of each block you need some metadata, with at least:

- block status (allocated / free)
- The block size



Traversal

Start by reading the metadata on the left (first block):

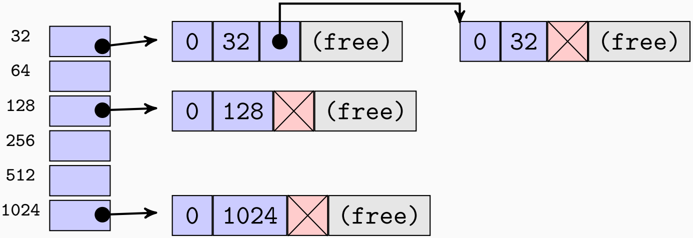
- If the block is not available, move the pointer from block's size bytes to check the next block.
- If the block is of the right size you just need to mark it as used.
- If the block is available but too big, it needs to be split:
    - Update its metadata (reduce block size).
    - Create new metadata for the buddy issued by the division.
- The buddy of a block can be easily accessed with the formula:
    - `block_addr ^ block_size`

Note: With the metadata, a block of size n (n a power of two) can fit for an allocation of size `n - sizeof (metadatas)`.

Improvements

- Allocation is very slow, while freeing a block is fast.
- It’s almost necessary to use a datastructure to keep track of the free blocks.
- Various datastructure might be used : sized free-lists, binary tree..

Example using sized free-lists



Conclusion

Questions

Any questions?