

Report for Memory allocation report

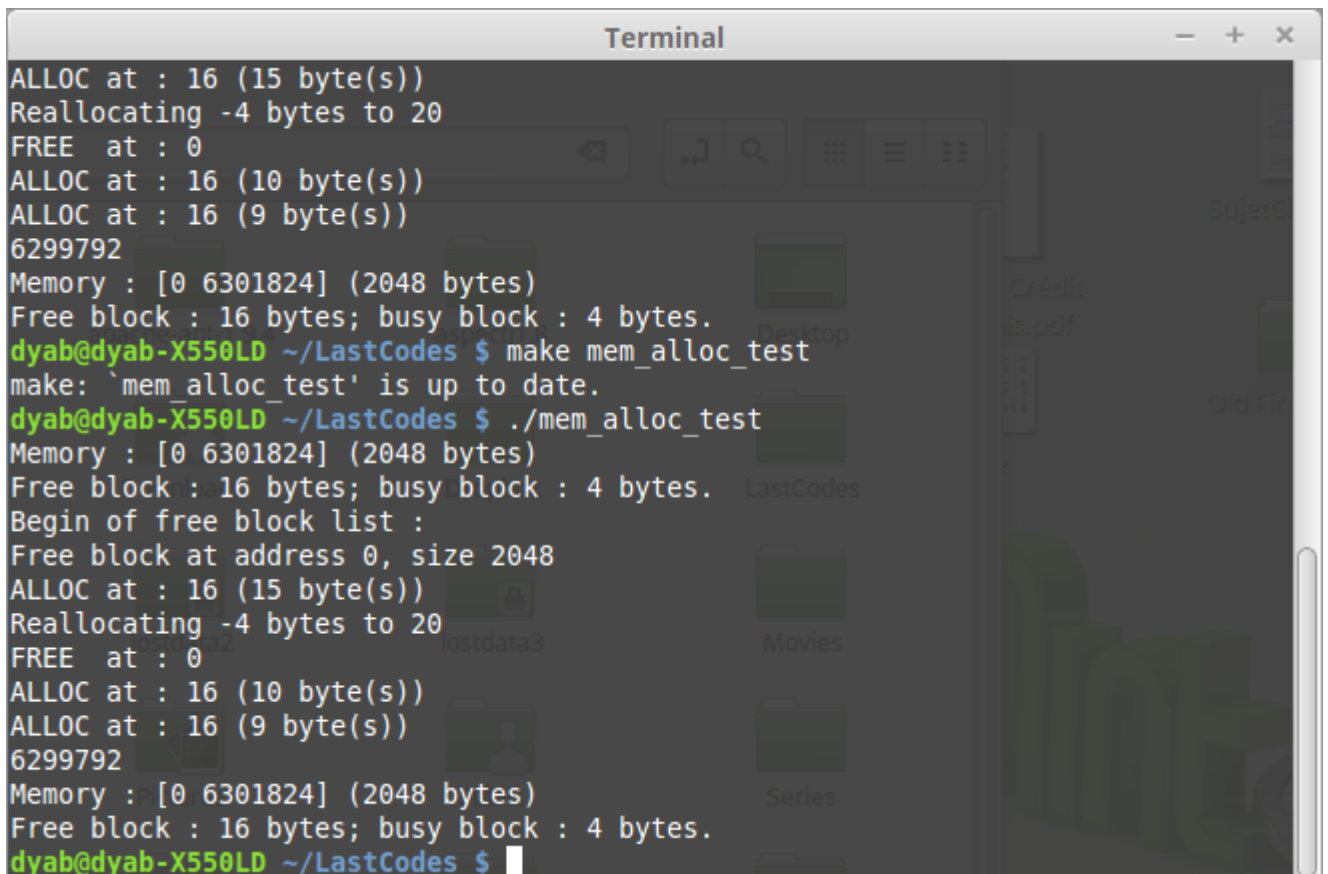
Ebrahim Naeimi - Olesia Vlassenkova

09/10/2015

This report includes information about memory management techniques and its implementations. We could design and implement the First-Fit policy.

Freeing memory of allocated blocks has a more complicated procedure depending on status of previous and next blocks.

A sample of executing the program for 3 amount of memory (15, 10 , 9 bytes)



```
Terminal
ALLOC at : 16 (15 byte(s))
Reallocating -4 bytes to 20
FREE at : 0
ALLOC at : 16 (10 byte(s))
ALLOC at : 16 (9 byte(s))
6299792
Memory : [0 6301824] (2048 bytes)
Free block : 16 bytes; busy block : 4 bytes.
dyab@dyab-X550LD ~/LastCodes $ make mem_alloc_test
make: `mem_alloc_test' is up to date.
dyab@dyab-X550LD ~/LastCodes $ ./mem_alloc_test
Memory : [0 6301824] (2048 bytes)
Free block : 16 bytes; busy block : 4 bytes.
Begin of free block list :
Free block at address 0, size 2048
ALLOC at : 16 (15 byte(s))
Reallocating -4 bytes to 20
FREE at : 0
ALLOC at : 16 (10 byte(s))
ALLOC at : 16 (9 byte(s))
6299792
Memory : [0 6301824] (2048 bytes)
Free block : 16 bytes; busy block : 4 bytes.
dyab@dyab-X550LD ~/LastCodes $
```

The code for memory allocation is as follow:

```
char *memory_alloc(int size){
    free_block_t pointer = first_free;
    //First Fit Policy
    while (pointer == NULL || pointer->size < size + sizeof(busy_block_s)) {
        pointer=pointer->next;
    }
    if(pointer == NULL){
        return NULL;
    }
    char *allocated_address = AllocateMemoryBlock((char*)pointer,size);
    //Worst Fit Policy
    pointer = FindWorstFit(size);
    AllocateMemoryBlock(pointer,size);
    //Best Fit Policy
    pointer = FindBestFit(size);
    AllocateMemoryBlock(pointer,size);
    print_alloc_info(allocated_address, size);
    return allocated_address;
}

char *AllocateMemoryBlock(char * pointer,int size) {
    free_block_t fb = (free_block_t)(pointer + size + sizeof(busy_block_s));
    //The size of new free block (after allocation) is equal to size of block which referred by "pointer" -
    //rerequested size + size of descriptor
    fb->size = ((free_block_t)pointer)->size - (size + sizeof(busy_block_s));
    //next block of new free block is the same as next of "pointer" block
    fb->next = ((free_block_t)pointer)->next;
    busy_block_t bb = (busy_block_t)pointer;
    bb->size = size + sizeof(busy_block_s);
    //Resetting next pointer of previous block
    ((free_block_t)pointer)->next = fb;
    return (char*)(bb + sizeof(busy_block_s)); //Returns the address of allocated block
}
```

The code for freeing Memory is as follow :

```
void memory_free(char *busyblock){
    free_block_t current_pointer = first_free;
    while (current_pointer != NULL) {
        if(current_pointer->next<=busyblock){// Checks left side blocks of busyblock for a freeblock
            if ((current_pointer + current_pointer->size) == busyblock) { //There is no any busyblock
between
                //if (current_pointer->next = busyblock + busyblock->size) { //This condition also can be used
                    current_pointer->size = current_pointer-> size + busyblock->size;
                    //current_pointer->next=busyblock+busyblock->size;
                }
            else { //here is a block between in the left side of busyblock
                free_block_s new_freeblock = busyblock + sizeof(busyblock->size);
                new_freeblock->size=busyblock->size;
                new_freeblock->next=current_pointer->next;
                current_pointer->next=busyblock;
            }
        }
        else { // Next freeblock is after the busy block
            if(current_pointer->next=(free_block_t)((char*)busyblock + busyblock->size) { //Next block
of busyblock is a freeblock
                busyblock->next=(free_block_t)((char*)busyblock + busyblock->size);
            }
            else { //There is a busyblock between
                current_pointer->next=busyblock;
                free_block_s new_freeblock = busyblock + sizeof(busyblock->size);
                new_freeblock->size=busyblock->size;
                new_freeblock->next=current_point->next;
            }
        }
        current_pointer = current_pointer->next;
    }
    print_free_info((char*)first_free);
}
```

Questions and answers

Question III.1: Do you have to keep a list for occupied blocks ? If no, explain why, if yes, provide a C definition of the structure.

This algorithm does not need to keep a list of busy blocks. Just by keeping list of free blocks, we are able to manage memory allocation for the requested blocks by the user. By using this list we can calculate previous and next busy blocks.

Question III.2: As a user, can you use addresses 0,1,2 ? Generally speaking, can a user manipulate any address?

User cannot use the beginning of memory addresses such as 0 and 1. Because these addresses are reserved for descriptor information of first block of memory.

Question III.3: When a block is allocated, which address should be returned to the user?

Beginning address of allocated block without considering descriptor address of the block, should be returned to the user.

Question III.4: When a block is freed by the user, which address is used? What should be done in order to reintegrate the zone in the list of free blocks?

When freeing a block by the user, the address of busy block is used for freeing the block and doing merge procedure to bring back the block to the memory space. Different policies should be applied depending on position of the busy-block and its neighbor blocks,

Question III.5: When a block is allocated inside a free memory zone, one must take care of how the memory is partitioned. In the most simple case, we allocate the beginning of the block for our needs, and the rest of it becomes a new free block. However, it is possible that the rest is too small. How is this possible ? How could you deal with this case?

In Best –Fit policy it is more likely that the rest of block becomes useless.

So, to deal with this problem there are some solutions. One is to integrate all free block together to make a big free block suitable to allocate to the other user requests. The policy of integration can be different depending on allocation policy.