HW #4

Memory Management Functions

Yunmin Go

School of CSEE

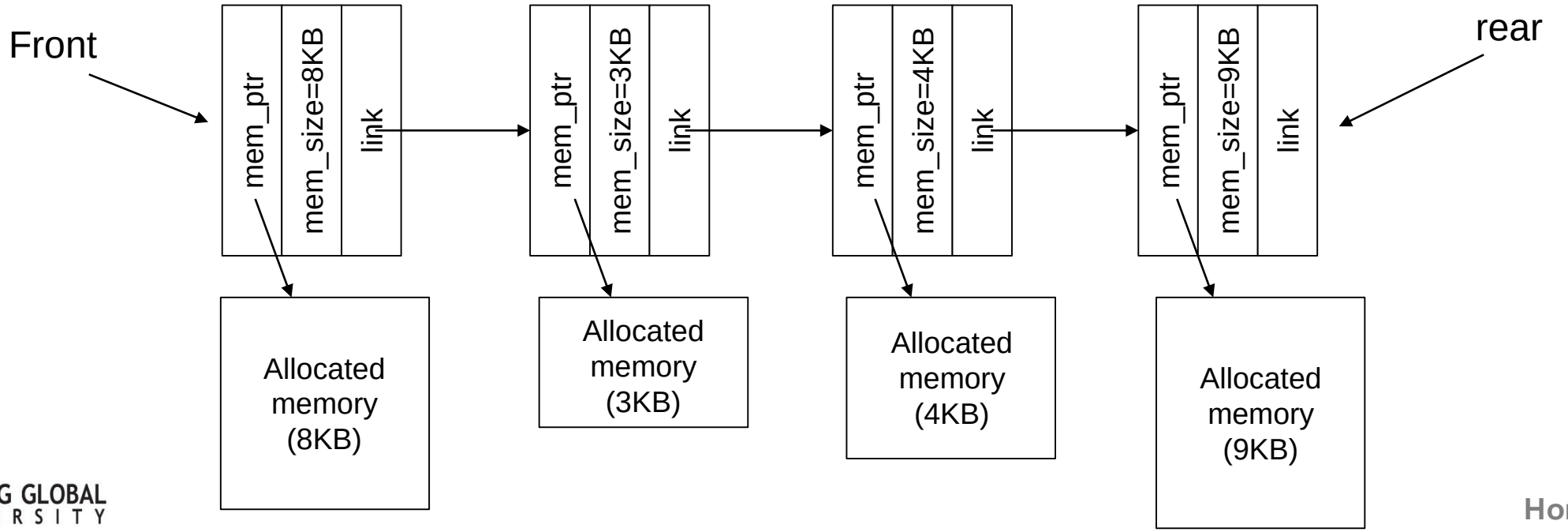
HANDONG GLOBAL UNIVERSITY

Requirements

- Implement memory management functions using system calls
  - For given `mm.h` and `mm_main.c`, you have to implement following functions in mm.c
    - `mm_malloc()`: similar to `malloc()`
    - `mm_calloc()`: similar to `calloc()`
    - `mm_realloc()`: similar to `realloc()`
    - `mm_free()`: similar to `free()`
    - `mm_status()`: print the status of allocated memory

Requirements

- Memory allocation list
  - Functions implemented in this homework manage a memory allocation list
  - Memory allocation list is implemented by linked list
  - Memory allocations list contains the memory information including the pointer and memory size of allocated memory



Requirements

- `mm_malloc()`
  - `void *mm_malloc(size_t size)`
  - Similar to `malloc()`
  - This function allocates *`size`* bytes and returns a pointer to the allocated memory
  - This function adds a memory information to the memory allocation list

Requirements

- `mm_calloc()`
  - `void *mm_calloc(size_t num, size_t size)`
  - Similar to `calloc()`
  - This function allocates memory for an array of *`num`* elements of *`size`* bytes each and returns a pointer to the allocated memory
  - This functions also adds memory information to the memory allocation list

Requirements

- `mm_realloc()`
  - `void *mm_realloc(void *ptr, size_t size)`
  - Similar to `realloc()`
  - This function changes the size of the memory block pointed to by *`ptr`* to *`size`* bytes
  - The contents will be unchanged in the range from the start of the region up to the minimum of the old and new sizes
  - If the new size is larger than the old size, the added memory will not be initialized.
  - This functions modifies the corresponding memory information in the memory allocation list

Requirements

- `mm_free()`
  - `void mm_free(void* ptr)`
  - Similar to `free()`
  - This function frees the memory space pointed to by *`ptr`*, which must have been returned by a previous call to `mm_malloc()`, `mm_calloc()`, or `mm_realloc()`
  - This functions removes corresponding memory information from the memory allocation list

Requirements

- `mm_status()`
  - `void mm_status()`
  - This function prints the address and size of allocated memory by traversing the memory allocation list
  - You can check the expected results and `mm_main.c`

Requirements

- You must use `mmap()`, `munmap()`, and `mremap()` for memory management
  - In fact, malloc() can be implemented using `brk()` or `sbrk()`. However, in this homework, we use `mmap()` for simplification.
- Don't use `malloc()`, `calloc()`, `realloc()`, and `free()` for this homework
- Functions should be implemented in `mm.c`
- You can add additional functions and files if you need
- You can modify `mm.h`
- Your program should be executed on Ubuntu
- For unmentioned requirements, you can implement freely.

Requirements

- Expected results

```
yunmin@mcnl-server:~/workspace/os/hw4$ ./mm_main
** Step #1 **
x + y = 30
array[0] = 0
array[1] = 1
array[2] = 4
array[3] = 9
array[4] = 16
array[5] = 25
array[6] = 36
array[7] = 49
array[8] = 64
array[9] = 81
Allocated Memory:
[  0] Addr=0x7f08c2e35000, Size=4
[  1] Addr=0x7f08c2e07000, Size=4
[  2] Addr=0x7f08c2e05000, Size=4
[  3] Addr=0x7f08c2e03000, Size=40

** Step #2 **
x + y = 60
Allocated Memory:
[  0] Addr=0x7f08c2e35000, Size=4
[  1] Addr=0x7f08c2e07000, Size=8
[  2] Addr=0x7f08c2e05000, Size=4
[  3] Addr=0x7f08c2e03000, Size=40
```

mm_status()

```
** Step #3 **
Allocated Memory:
[  0] Addr=0x7f08c2e35000, Size=4
[  1] Addr=0x7f08c2e05000, Size=4
[  2] Addr=0x7f08c2e03000, Size=40

** Step #4 **
array[0] = 0
array[1] = 1
array[2] = 4
array[3] = 9
array[4] = 16
array[5] = 25
array[6] = 36
array[7] = 49
array[8] = 64
array[9] = 81
array[10] = 100
array[11] = 121
Allocated Memory:
[  0] Addr=0x7f08c2e35000, Size=4
[  1] Addr=0x7f08c2e05000, Size=4
[  2] Addr=0x7f08c2e03000, Size=48
```

Requirements

- Write clean source code
  - Add proper comment in your source code
  - Consider code indentation for enhancing readability
- Test your source codes with many cases for self verification
- Upload tar.gz file on LMS by compressing all your source codes
  - Includes `mm.c`, `mm.h`, and `mm_main.c` (plus your additional files)
  - File name: hw04_student id.tar.gz (ex: hw04_20400022.tar.gz)

- Due date: 11:59pm, 6/21 (Tue)