CS2106: Operating Systems
Lab 4 – Buddy System Simulation

Important:
- The deadline of submission on IVLE is **23rd October 5pm**
- The total weightage is 6%:
  - Exercise 1: 1% [Demo Exercise]
  - Exercise 2: 2%
  - Exercise 3: 3%
- System/OS restriction: Use **any Unix based** OS for this lab. You can either use **Sunfire** or any **Linux** installation.

Section 1. Overview

Since memory management is performed by the Operating System internally, it is not easy to understand the actual process from outside. Hence, we will write a **simulation** of memory allocation to have a better understanding of the process in this lab.

There are three exercises in this lab, which are all based on **Buddy System** dynamic memory allocation scheme. The exercises essentially break the buddy system down to the following:

- Exercise 1: Setting up the array of linked lists based on the initial memory size.
- Exercise 2: Handling allocation and splitting.
- Exercise 3: Handling deallocation and merging.

Note that the exercises are **not independent**. Rather the solution from the earlier exercise should be reused and built upon for later exercises.

Section 2. Exercises in Lab 4

Please refer to the lecture note for the buddy system description and algorithms. A couple of implementation tips are suggested in the respective sections.

2.1 Exercise 1 – Setup and Debug

In this exercise, your main job is to:
1. Setup the array A[] according to the initial memory size. You can assume that the initial memory size is no more than 4096. The memory size is **always a power-of-2.**
2. Implement a printing function that print out the array A[] to support debugging in exercise 2 and 3.

Sample Session (user input in **bold**):	
256	
A[8]: [0]	*//1 free block of 2^8 (256) at address 0*
A[7]: EMPTY	*//no free blocks on any other levels*
A[6]: EMPTY	
A[5]: EMPTY	
A[4]: EMPTY	
A[3]: EMPTY	
A[2]: EMPTY	
A[1]: EMPTY	
A[0]: EMPTY	

The debug printing function should print the linked list at each level of the array. For non empty linked list, the starting address of each free block is printed.

Note: This exercise is really just for you to code the basic structure and debugging function. Your lab TA will briefly inspect your code to ensure you have done something useful (rather than just "faking" the printout ☺).

2.2 Exercise 2 – Allocation and Splitting

In this exercise, you need to handle the memory allocation request. Modify your solution from ex1 to accommodate the following changes:

1. Ask for initial memory size.
2. **Ask for total number of request, T.**
3. **Read T requests, each request has the format:**
   "1 *request_size*"

   Note: the "1" is to distinguish between the allocation and deallocation request. It is only useful in ex3.
4. **Allocate a block suitable for *request_size* using buddy system.**
   a. **Report the allocated block starting address if successful.**
   b. **Print a "-1" if the block cannot be allocated.**

Sample Session **Input**	**Output**
256 *//initial memory size* **4** *//4 requests* **1 64** *//request to allocate 64 bytes* **1 30** **1 120** **1 128**	0 //block allocated at address 0 64 //block allocated at address 64 128 -1 //failed to allocate

To ease your programming effort, you are given "testX.debug" in the ex2/ directory which contains detailed debug tracing information.

Note: Make a copy of your ex1.c and rename it ex2.c, then modify and add on.
Note 2: Make sure you turn off the debug output for your submission.

Hint 1: Don't be afraid of using a little recursion ☺.
Hint 2: Although it is not required for this exercise, you are strongly encouraged to keep the successful allocation information in some fashion, minimally keeping track of the {starting address, allocated block size, actual size in use} so that you'll have an easier time for ex3.

2.3 Exercise 3 – Deallocation and Merging

You are now ready to complete the entire Buddy System implementation. Modify your implementation from ex2 to handle memory deallocation requests. The modified steps are in **bold**:

1. Ask for initial memory size.
2. Ask for total number of request, T.
3. Read T request, each request has the format:
   "1 *request_size*" for allocation requests OR
   "2 *starting_address*" for deallocation requests.

4. For allocation request: Allocate a block suitable for *request_size* using buddy system.
   a. Report the allocated block starting address if successful.
   b. Print a "-1" if the block cannot be allocated.

5. **For deallocation request: Free the block at the indicated *starting_address***
   a. **Print "ok" if the block can be freed.**
   b. **Print "failed" if the block cannot be found (i.e. the block starting address is incorrect.**

6. **After all requests have been performed. Print the following 3 statistics, 1 on each line:**
   a. **Total memory in use (allocated).**
   b. **Total free memory (i.e. a + b = initial memory size).**
   c. **Total internal fragmentation (for each allocated block, sum the allocated size – actual requested size).**

Sample Session **Input**	**Output**
256 *//initial memory size* **6** *//6 requests* **1 60** **1 62** **1 64** **2 64** *//free block at address 64* **2 128** *//free block at address 128* **2 64** *//free block at address 64*	0 //64 bytes allocated at address 0 64 //64 bytes allocated at address 64 128 //64 bytes allocated at address 128 ok //64 bytes at address 64 freed ok //64 bytes at address 128 freed failed //failed to free (block not allocated) 64 //statistic a: 64 bytes in use 192 //statistic b: 192 bytes free 4 //statistic c: 4 bytes wastage // 60 bytes of the 64 bytes at address // 0 is in use, 4 bytes wasted

Note: Make a copy of your ex2.c and rename it ex3.c, then modify and add on.

Hint 1: You need to keep track of the allocated memory block in order to handle deallocation and the statistics printing.
Hint 2: Similar to allocation, the free and merge operation is quite suitable to a recursive solution. Not compulsory, but definitely makes your code shorter ☺.

Section 3. Submission

Zip the following files as A0123456.zip (**use your student id!**):

a. ex2.c
b. ex3.c

Upload the zip file to the "Student Submission→Lab 4" workbin folder on IVLE. Note the deadline for the submission is **23rd October, 5pm**.

Again, please ensure you follow the instructions carefully (output format, how to zip the files etc). Deviations will cause unnecessary delays in the marking of your submission.