CS2106 Lab 4 - Part 1

Exercise 1: Synchronisation again?!

This is the lab demo exercise. Your code and output should be shown to your lab TA in week 10 or week 11. You should submit your code to LumiNUS (lab4-demo-submissions folder) by March 27, 10PM. All four files should be zipped and uploaded as <NUSNET_ID>.zip (e.g., E0123456.zip), without any folder structures.

After the CS2106 Midterms, Ralph feels confident that he can solve any synchronisation problem. He loves semaphores and is eager to show off their prowess to his other friends.

Ralph designs a new algorithm to sort an array of N numbers (that also somehow uses semaphores). He bases this off this known algorithm known as `sleep sort` (look it up!):

```
1. Start N threads and run them for each element in array
2. Each thread will sleep for a time proportional to its value and then
calls wait() to get blocked
3. Notify the waiting threads one by one
4. Each notified thread add their value to a result array - from the back
(e.g. if thread 1 and 2 are woken up in this order, the array will be [2,
1])
5. Return result array - it should be sorted!
```

He uses a binary semaphore to implement the blocking and signalling mechanisms. He tested on some cases (N = 1 and N = 0) and found that it works!

However, on the day he was supposed to show his friends, he realised his code is completely wrong. He realised he made some assumptions about which process is woken up when a `signal` is done.

Your task is to modify Ralph's code to such that the program correctly sorts the array. You should not touch his algorithm logic and only rewrite the functions in `ex1.c`. You can add in more variables (if you need) in `custom_lock.h`.

Hint: You notice Ralph is putting the numbers from the back. Thus, if thread 5 wakes up last, it will be the first element in the array. The ordering of who is woken up when we do `sem_post` depends on the OS. You should implement a semaphore with an appropriate semantics of the signal operation such that Ralph's code works.

For this lab you are provided with four files:

- `ex1_runner.c`: runs your code and prints the relevant results.
- `my_semaphore.h`: function definitions file
- `custom_lock.h`: a struct definition file for your convenience
- `ex1.c`: you will find the functions you need to implement here

You are only **supposed to modify custom_lock.h and ex1.c**. The other files are simply to run and check your code.

To compile and run the program, do:

```
gcc -Wall -std=c99 ex1_runner.c ex1.c -o ex1 -lpthread
./ex1 <N>
```

<N> is a number you must pass in. It is the size of the array created.

The program will:

```
1. generate an array of numbers, [1,2,3,4... N]
2. randomly shuffle them
3. try ralph's algorithm
4. print result array and check if it is sorted
```

You have to fix the implementation of the functions defined in `ex1.c` such that Ralph's algorithm still works.

Note: It is recommended to use the `xcnd` cluster servers for this lab. All provided code is tested in that environment. Certain functionalities (e.g. semaphores) may not work correctly on your system.

To get the full mark for this lab, you have to:

- sort the array (correctness)
- make it work for reasonably sized N (0 <= N <= 200)
- use semaphores, Ralph loves semaphores remember?