Assignment 3: Problems in Synchronization

Aim: To understand threads and synchronization using pthread library

Prerequisite: You must have a good grasp of how threads work and how to use locks and conditional variables.

Background Reading:

1. https://randu.org/tutorials/threads/
2. https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html

Problem 1: Hare and Tortoise:

You will have to write a few cooperating threads -

1. Turtle
2. Hare
3. God
4. Reporter

As we know, since our childhood, hare and turtle went for a race, and turtle wins !! why? maybe God was on its side. Now, let's recreate the scene and assert that it's the God who decides the winner. We have four entities who are working in this scenario,

```
Turtle () {
        while (race_not_ends) {
                move_leg(1);
                move_leg(2);
                move_leg(3);
                move_leg(4);
        }
}

Hare () {
        while (turtle is far off)
                Sleep (for some time);
        Run_like_crazy (for some time);
}

God () {
         Select hare or turtle and reposition;
}

Reporter () {
        while (race is on) {
                Report the positions of hare and turtle in distance from the start;
        } }
```

Init () {

Create 4 parallel threads and run each of the four processes (Hare / Turtle / God and Reporter.

while (race is on);

}

Develop the application (hare and turtle) using Linux pthreads (shared memory). Notice that you have shared memory between threads so you do not need explicit communication channels. You can communicate using shared variables. However, you will need to ensure that different threads synchronize their access to shared variables to avoid inconsistencies. In particular, you should use mutexes(locks) and conditional variables (see tutorial) to protect all accesses to shared variables.

Tests:

- The current outcome of the race should be displayed after fixed intervals.
- Make sure to make a critical section as small as possible.
- Try varying the speed and the sleeping time of the hare to see different outcomes.
- Few test files are provided in the TestCase folder.
- Passing these test cases doesn't mean the solution is correct. Your solution will be tested against hidden test cases. Check for corner cases.
- Compile using **make**
- Run using **./hare_tortoise test-file=<path to file>**
- Make sure there are no spaces in the file path.

Marks distribution:

- 10 marks for Init function.
    - You are required to create threads and make sure all threads have been terminated before exiting the function.
    - You might want to initialize lock, cv and variables that you add to a struct.
- 10 marks for turtle function.
    - You can simply use delay/sleep and update distance covered per millisecond.
- 10 marks for hare function.
- 10 marks for god function.
- 10 marks for result printing.
- Negative marking:
    - -5 if turtle/hare/god/result is still running after the exit of Init function.
    - -3 per function if the mutual exclusion is not guaranteed.
    - -2 per function if the deadlock can happen
    - -2 per function if progress for all functions is not guaranteed i.e. if one function holds the lock and never releases it causing other processes to stop.

Problem 2: Hydrogen oxygen reaction:

You have been hired by Mother Nature to help her out with the chemical reaction to form water, which she doesn't seem to be able to get right due to synchronization problems. The trick is to get two H atoms and one O atom together at the same time. Each atom is represented by a thread.
Each H atom invokes the function void reaction_h(struct reaction *r) when it is ready to react, and each O atom invokes the function void reaction_o(struct reaction *r)

You must write the code for these two functions. The functions must delay until there are at least two H atoms and one O atom present, and then exactly one of the functions must call the procedure make_water. After each make_water call, two instances of reaction_h and one instance of reaction_o should return.
Create a file reaction.c that contains the functions reaction_h and reaction_o, along with a declaration for struct reaction (which contains all the variables needed to synchronize properly).

In addition:

- Write the function:
  reaction_init(struct reaction *r)
  which will be invoked to initialize the reaction object, locks and CVs.
- Your code must invoke ***make_water*** exactly once for every two ***H*** and one ***O*** atoms that call ***reaction_hreaction_o***, and only when these calls are active (i.e. the functions have been invoked but have not yet returned).

Marks distribution:

- 10 marks for ***Init*** function.
  - You might want to initialize lock, cv and variables that you add to a struct.
- 20 marks for ***reaction_h*** function.
- 20 marks for ***reaction_o*** function.
- Negative marking applies for only for ***reaction_h*** and ***reaction_o***:
  - -6 per function if the mutual exclusion is not guaranteed.
  - -4 per function if the deadlock can happen
  - -4 per function if progress for all functions is not guaranteed i.e. if one function holds the lock and never releases it causing other processes to stop.

Note: For both problems,

- Write your solution in C using the cs_thread.h functions for locks condition variables ONLY!!
- Your code must not result in busy-waiting.

We have provided a template for the reaction.c and hare_tortoise.c files - just fill in the functions. We have also provided a Makefile and a test harness for a reaction.c to run some simple tests. You need to submit back the tarball with the reaction.c and hare_tortoise.c function templates filled in.
DO NOT CHANGE THE TESTS OR THE MAKEFILE.

Submission instructions:

You have to submit just 2 files, reaction.c and hare_tortoise.c. Feel free to change test cases to see how your program behaves at corner cases. Avoid modifications to other files. Put mentioned 2 files in a folder with your roll numbers as a name. Compress it and submit. eg. for roll numbers 183050011 and 183050012 folders will be 183050011_183050012. Then create a tar.gz file using :

tar -cvzf 183050011_183050012.tar.gz 183050011_183050012/

How to run code:

1. Use **make** to build.
2. Use **make run** to run it.
3. Use **make clean** to clean project.

I will create a new thread for Assignment 3. Feel free to ask any queries there. Have fun.