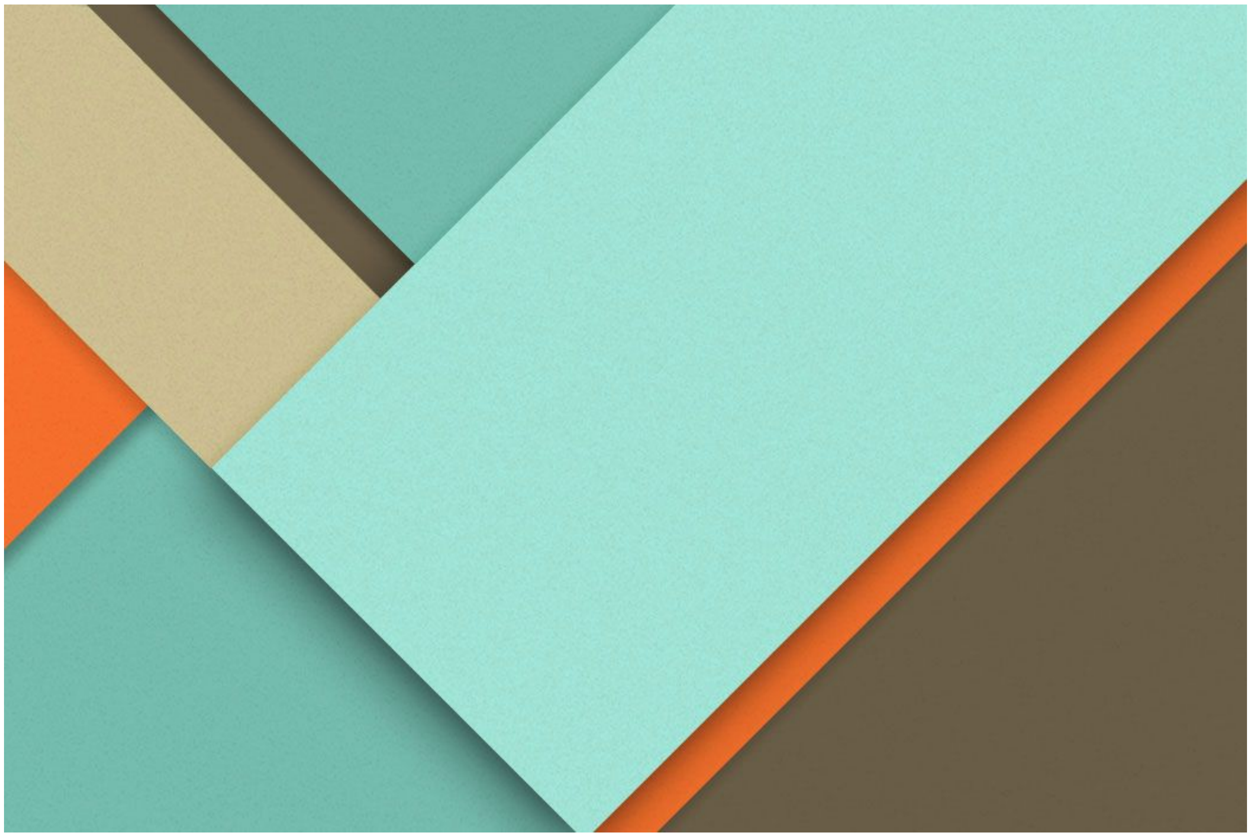

Hare and Tortoise Race

13.02.2019

Kuldeep Singh Bhandari
111601009

CSE, 3rd Year Btech

IIT PALAKKAD

DESCRIPTION :

- The program is for LINUX Operating System.
- Run program using command :
  - **make**
  - **./driver.out**
- Global variables :
  - **tortoise (long) :** distance counter for tortoise
  - **hare (long) :** distance counter for hare
  - **tortoise_time (long) :** time counter for tortoise
  - **hare_time (long) :** time counter for tortoise
  - **TARGET (const long) :** target value to finish the race
  - **STEPS (const long) :** steps taken by the hare in one iteration
  - **MIN_DIST_FOR_SLEEP (const long) :** min. amount of distance should hare and turtle have between them so that hare can sleep
  - **tortoise_mutex (pthread_mutex_t) :** mutex to access tortoise variable
  - **hare_mutex (pthread_mutex_t) :** mutex to access hare variable
  - **terminal_mutex (pthread_mutex_t) :** mutex to access terminal
- This is the main program which creates four threads :
  - **Tortoise** : Tortoise thread runs until tortoise has not reached the target.
  - **Hare :** Hare thread runs until hare has not reached the target.
  - **Reporter :** Reporter thread runs until either of tortoise or hare has not reached the target.
  - **God** : God thread runs until either of tortoise or hare has not reached the target.
- There are three mutexes used in this program.
- Operations of threads 'tortoise', 'hare' and 'reporter' are parallel as there is no mutex between them.
- When God comes into play, God halts the other three threads using mutexes :

   - ➢ God halts hare and tortoise threads so that "tortoise and hare do not reach to the end while God is making decision".
   - ➢ God halts reporter processes so that "reporter do not report in between when god is making decision"
- ❖ Parent process waits for the all the four threads to complete and then it declares who is the winner.
- ❖ step for tortoise = 1, step for hare = STEPS

APPROACH

The initial idea was that we will allow hare and tortoise to run in parallel and increment their positions depending upon the steps they take and declare the winner as soon as either of them reaches the target position. The issue with this approach is that it may be possible that one process is being scheduled more than the other by Operating System and because of that, one process will be allowed to run for more time than the other. So, it is even possible that one thread ends before the other has started. Declaring winner with this measure would mean that the process which is being scheduled for more time is the winner, not the one which has reached the destination in the minimum time. So, this approach is dropped.

To handle the issue with the previous idea, we can introduce two time counters for hare and tortoise. So, now when tortoise and hare are allowed to take their steps and we will increment hare time counter and tortoise time counter by 1. What this will do is it will keep track of “for how many iterations hare and tortoise has run before reaching to the target”. So, now we will allow hare and tortoise to run in parallel and wait for both of them to finish their race and finally, see who has taken less time to reach to the target. The positive aspect of this approach is that we now don’t care about how operating system is scheduling the threads.

God can change the tortoise and hare positions randomly only if one has not reached to the target. Once one has reached to the target, God will not alter its position.