Simulate hare and turtoise run using threads and mutexes

Solution:

There are five shared variable:

1. hareDist: postition of hare
2. turtoiseDist: position of turtoise.
3. hareTime: Time taken by hare.
4. turtoiseTime: Time taken by turtoise.
5. minIntervalRequired: If hare is ahead of turtoise at least by this distance then it can sleep for random interval.
6. minHareSleeps: Minimum number of iteration hare sleeps if it's far ahead of turtoise.

However among thes five shared variable only two harDist and turtoiseDist are shared accessed by multiple threads and hence we need to ensure the mutual exclusion for these variables.

Fuction Description:

1. hareFunction: Simulates running of hare. It proceeds in following order.
    1. If hare is sleeping update time of hare and **continue**.
    2. Check if hare is far ahead of turtoise.
    3. If hare is far ahead of turtoise sleep for random number of iteration.
    4. Else update position of hare.
    5. Update time of hare.
2. turtoiseFunction: Simulates running of turtoise. it updates poistion and time of turtoise.
3. reporterFunction: Prints position of hare and turtoise on terminal.
4. godFunction: Gets new position of hare and turtoise and updates respectively and then sleep for 500 microseconds.
5. getNewVal: It's a helper function for god. It ouputs new random position of hare and turtoise.
6. createThread: A utility function to create thread and exit with error if thread can not be created.

Approach for Mutual Exclusion:

- We have two shared variable hareDist and turtoiseDist. Each access to these shared variable should be protected.
- However we will not ensure the mutual exclusion when reporter is reporting.
- We have two mutexes hareDistMtx and turtoiseDistMtx to protect the accesses to hareDist and trutoiseDist shared variable respectively.

Procedure:

1. If turtoise wants to update its position it first locks turtoiseDistMtx mutex, then updates turtoiseDist and then unlocks the turtoiseDist Mtxt.
2. Hare needs to have access to its own distance and turtoise distance both because it sleeps if it is far ahead of turtoise. Hare function proceeds in following order.
    - Lock hareDistMtx and turtoiseDistMtx in order.
    - Either updated hareDist or decides to sleep.
    - Unlock turtoiseDistMtx and hareDistMtx in order.
3. As Discussed we don't worry about mutual exclusion when reporter access the shared variables.
4. God needs to update both hareDist and turtoiseDist depending on the input. God function proceeds in following order.
    - Lock hareDistMtx and turtoiseDistMtx in order.
    - Update hareDist and turtoiseDist depending on the input.
    - Unlock hareDistMtx and turtoiseDistMtx.

DeadLock: Is there any chances of deadlock. Let's go through four required condition for deadlock.

1. **Mutual Exlusion**: Clearly this condition is satisfied because hareDistMtx and turtoiseDistMtx(or equivalently hareDist and turtoiseDist) are held in nonsharable mode because only one thread can lock(or access) them at a time.
2. **Hold and Wait**: One of the hold and wait situation is when hare thread locks hareDistMtx and then tries to lock the turtoiseDistMtx. Hence Hold and Wait Condition is satified.
3. **No preemption**: It's satisied because of the propety of mutex "only thread which locks the mutex can unlok it".
4. **Circular Wait**: Notice that **Hold and Wait** condition occurs only when hare thread(or God thread) lock hareDistMtx and then tries to lock the turtoiseDistMtxt. But They do that is order i.e. they first lock hareDistMtx and then only try to lock turtoiseDistMtx. Due to this ordering constraint Circular Wait condition is not satisfied and hence **Deadlock cannot occur**.