1. Intro to synchronization mechanisms

– Discussion about: mutex, lock_guard, event handling (condition variables), future/promise, async.

- Read tutorials 4-9 from here: https://thispointer.com/c11-multithreading-part-4-data-sharing-and-race-conditions/ AND write examples on your machines

2. Exercise: implement a Semaphore - missing from Standard. (Solution is in MySempahore.hpp) ;

   Live explanations.

3. Readers & Writers problem: https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem

   Motivation :
   - very important topic in industry
   - it gives you the opportunity to understand how to build a simulation.
   - high difficulty

Live discussion and solutions:
- Look inside ReadersAndWriters.cpp and Simulation.cpp to understand the big picture
- Each implementation is done using template policies.

3. A Simple attempt 1
   No Locks ?

3.B Priority for readers
   Check ReadersAndWriters_ReadPriority.hpp

3.C Priority for writers
   Check ReadersAndWriters_WritersPriority.hpp

3.D Fairness for both sides using a fair semaphore.
   - Look at ReadersAndWriters_Fair to see how it is used.
   - Try to implement YOURSELF the fair semaphore derived from MySemapohore (solution is in MyFairSemaphore.hpp code).
   - Idea: we are using a queue of waiting threads. On wait() func implementation, block on a condition variable/mutex and register its personal condition variable/mutex in the queue. On signal() take the first thread waiting in

the queue and call notify_one/unlock on its registered condition variable.

==========

4. Debugging

Each IDE has a mechanism to debug thread issue:
- For difficult problems (deadlocks for example) I suggest VS and its Parallel Stack view:
https://docs.microsoft.com/en-us/visualstudio/debugger/using-the-parallel-stacks-window?view=vs-2017

- Tutorial for debugging threads in VS 2017: https://docs.microsoft.com/en-us/visualstudio/debugger/get-started-debugging-multithreaded-apps?view=vs-2017

- All other have at least suspend and call stack thread investigation (so don't use command line to debug parallelism).

5. Other things to consider for your projects:

5.1 Better testing

Build a test check if the output is correct.
How would you do that ?

5.2 Profiling - in the end you NEED profile results write in a graphical way like a file, table, image, etc: e.g. number of entities, parameters used for simulation, time needed to execute ,etc.

We simulated execution time using Sleep which is correct somehow for simulations. Just be sure that:
- You have parameters exposed in a ini/json/yaml etc to expose such parameters
- Read the number of writers / readers, numbers of cycles etc

5.3 Organize the code in a very flexible way: multiple compile units, clear connection between components, watch for memory issues specially when using MPI.

Last Warning: Sometimes system functions such as sleep or console write/read can make your threads synchronize. Don’t worry, but consider this when you debugging apparent lack of parallelism.