Project 2: Thread Synchronization - part (a)

Semaphores - Building H_2O_4S Molecules

This is an individual project. You will use a standard implementation of semaphores to synchronize threads representing atoms in forming molecules.

deliverables: your solution in the file H2SO4.c - to be compiled and run using the test code I provide in H2SO4Test.c. Make sure your solution works with that test method, and use good coding standard in general.

For this question, I have provided you with a header file in H2SO4.h that you may NOT modify and you MUST include in your solution program. You can think of this .h file as similar to an interface in Java - you must implement all of the functions in the .h file in your solution program, using the function names, parameters, and return types exactly as given. You should name your solution program H2SO4.c. You can look at (and use any part of) the provided example in exampleH2O.c which uses the same H2SO4.h file.

To compile the overall program, you simply list both .c files after gcc, e.g. “gcc H2SO4.c H2SO4Test.c –o molecules”, and make sure both .c files and the .h are all in the current directory. To run a test, include 3 integers as command line arguments, the number of hydrogen, sulfur, and oxygen atoms in that order, e.g. "./molecules 2 1 4" should create 1 full molecule. You can run a test of exampleH2O.c in the same way, just put a 0 for the sulfur argument.
You don’t need to modify the test program (though you may want to in order to test more thoroughly), and in fact you will only turn in your solution .c file, but you should check out the code given in the tests, as there’s lots of useful stuff, like command line arguments, and creating pthreads, and more!

You will use only **semaphores** as the synchronization methods for this problem. See exampleH2O.c for the 3 semaphore methods you can use (sem_open, sem_wait, and sem_post). You should not use any other methods from semaphore.h, as they may or may not be supported depending on your OS (e.g. sem_init or sem_getvalue). Remember you can use a semaphore as a basic lock for mutual exclusion as well as counting semaphores.

A molecule of sulfuric acid is composed of two hydrogen atoms, one sulfur atom, and four oxygen atoms. In this project, you will be writing the synchronization needed for three different types of threads to interact in a manner “similar” to how these atoms interact to form a molecule of sulfuric acid. Each thread type corresponds to one of the 3 atoms and the program for each atom type should go in the function with the same name. The very first thing you should do in each of the 3 functions is to output that this atom has been produced. After that the algorithm for each atom thread may differ, but the general idea will be to use semaphores to synch the atoms into forming molecules as described below. You may create as many semaphores as you wish (I use several in my solution), and should declare these as global variables so all threads can access them. You should open all used semaphores in openSems and close/unlink them all in closeSems.

In order to form a molecule of H2SO4 two hydrogen threads, one sulfur thread, and four oxygen threads must be present. If any are missing, the remaining ones **must wait** (efficiently, no busy waiting) until the missing threads arrive at some later time.

As each new thread arrives (is created), your program should check to see whether there are now enough threads to form an H2SO4 group. If some required atoms are still missing, the newly arriving thread should be blocked on some semaphore's wait queue. There is no limit to the number of threads that might be waiting (e.g. you might produce 100 Hydrogen atoms, all of which will have wait, before you produce any Oxygen or Sulfur atoms).

If all required atoms are present, you should PRINT that a molecule has been formed, and allow all threads in the molecule to depart, again PRINTING when each one leaves. The "molecule formed" print must come before all of the "leaving" prints, and there is an additional requirement: when a group of threads departs, the hydrogen threads must leave first, followed by the sulfur thread, followed by the oxygen threads. However, if there are more than the required number of threads of the same type waiting, it makes no difference which of the waiting ones is unblocked. Note that new threads may arrive and print their "produced" statements interleaved with the "leaving" statements of the just-formed molecule, however you should not allow the creation of the next molecule until all current atoms have left, even if all necessary atoms are present.

Look at the provided example output and format your output in the same way, using the same wording.

If enough atoms are present to create a molecule, the molecule must be created. It is acceptable for the last required atom that arrives to block temporarily, but you must guarantee that you will never wait indefinitely if all of the required atoms have been produced. Your program MAY hang indefinitely at the end if extra atoms are waiting that were not part of any created molecule, but they are not sufficient to create another molecule.

The sample program exampleH2O.c would not meet the requirements for this project in forming H2O molecules. Even in simplest test where have exactly the correct atoms to create one molecule (2 hydrogen, 1 oxygen), the hydrogen atoms do not wait for a whole molecule to be completed before exiting, and so will likely output "hydrogen exited" before the required "molecule produced" message outputs.

More subtly, this implementation could hang indefinitely without creating a possible molecule given the following sequence of events:

-- oxygen produced, waits on hydro_sem
-- oxygen produced, also waits on hydro_sem
-- hydrogen produced, signals hydro_sem, first oxygen wakes but promptly executes 2nd wait and goes back to sleep on hydro_sem
-- hydrogen produced, signals hydro_sem, second oxygen wakes but promptly executes 2nd wait and goes back to sleep on hydro_sem

This leaves the program hung without ever producing a molecule, even though the required 2 hydrogen and 1 oxygen have been produced. Basically, it is stuck with 2 partial molecules, each with 1 H and 1 O. If you produced 2 more hydrogen atoms, this would then successfully finish 2 molecules, but the requirement is that it produce the first complete molecule even if enough hydrogens to make the 2nd never arrive.