Assignment 2: Problems in Synchronization

Problem 1: Metro Loading

The Mumbai Metro has decided to improve its efficiency by automating not just its trains but also its passengers!! From now on, passengers will be robots. Each robot and each train is controlled by a thread. You have been hired to write synchronization functions that will guarantee orderly loading of trains. You must define a structure **`struct station`**, plus several functions described below.

When a train arrives at the station and has opened its doors, it invokes the function

`station_load_train(struct station *station, int count)`

where count indicates how many seats are available on the train - assume that no standing is allowed on these trains. The function must not return until the train is satisfactorily loaded (all passengers are in their seats, and either the train is full or all waiting passengers have boarded).

When a passenger robot arrives in a station, it first invokes the function

`station_wait_for_train(struct station *station)`

This function must not return until a train is in the station (i.e., a call to **`load_train`** is in progress) and there are enough free seats on the train for this passenger to board. Once this function returns, the passenger robot will move the passenger on board the train and into a seat (you do not need to worry about how this mechanism works). Once the passenger has boarded, it will call the function

`station_on_board(struct station *station)` to let the train know that it's on board.

Create a file the **`metro.c`** that contains a declaration for **`struct station`** and defines the three functions above, plus the function **`station_init`**, which will be invoked to initialize the station object when the Metro runs. In addition:

You must write your solution in C using the pthreads functions for locks and condition variables. Use only these functions (e.g., no semaphores or other synchronization primitives). ***For safety, we have wrapped pthreads library calls with some assertions in cs744_thread.h that is provided in the attached tarball. Use only the functions defined in that file. No direct pthread library call use is allowed.***

- You may not use more than a single lock in each **`struct station`**.
- You may assume that there is never more than one train in the station at once, and that all trains (and all passengers) are going to the same destination (i.e. any passenger can board any train).
- Your code must allow multiple passengers to board simultaneously (it must be possible for several passengers to have called **`wait_for_train`**, and for that

function to have returned for each of the passengers, before any of the passengers calls **`station_on_board`**).

- ***Your code must not result in busy-waiting.***

Compiling and Testing

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

Problem 2: Making Water

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.
- 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).
- Write your solution in C using the cs744_thread.h functions for locks condition variables ONLY!!
- You may not use more than a single lock in each struct reaction.
- Your code must not result in busy-waiting.

Compiling and Testing

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