CMPU-334 Threading Lab

Assigned: Monday, Feb 22nd
Due: Wednesday, Mar 2nd 11:29pm
Note: this problem set should be done individually, not in teams. The teams will apply to the three Pintos projects.

Problem 1: Train Automation

For this problem we will model a subway station using treads. Your task is to write synchronization functions that will guarantee an orderly loading of trains. Each passenger and each train is controlled by a thread. You must define a structure `struct station`, plus several functions described below.

When a train arrives in 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. 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 thread 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 `station_load_train` is in progress) and there are enough free seats on the train for this passenger to sit down. Once this function returns, the passenger boards the train and into a seat (you do not need to worry about how this mechanism works). Once the passenger is seated, the passenger thread will call the function

```
station_on_board(struct station *station)
```

to let the train know that they are on board.

Create a file `train.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 Train boots. In addition:

- You must write your solution in C using the Pintos functions for locks and condition

variables:

```
lock_init (struct lock *lock)
lock_acquire(struct lock *lock)
lock_release(struct lock *lock)
cond_init(struct condition *cond)
cond_wait(struct condition *cond, struct lock *lock)
cond_signal(struct condition *cond, struct lock *lock)
cond_broadcast(struct condition *cond, struct lock *lock)
```

Use only these functions (e.g., no semaphores or other synchronization primitives).

- 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 `station_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.

Problem 2: Chemical Reaction

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` (which you needn't write; Mother Nature's already gotten this part figured out). 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_h` `reaction_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 Pintos functions for locks and condition variables listed in Problem 1 above.

- You may not use more than a single lock in each `struct reaction`.

- Your code must not result in busy-waiting.

Testing Your Code

We have created a test framework that you can use to test your code.

```
make run
```

Will compile and test your solutions for both problems. Read the README file in that directory for instructions on how to use the testing framework.

Grading

This lab is worth 100 points. You will receive 40 points for passing the tests for problem 1 and 30 points for passing the tests for problem 2. Synchronization code is hard to get right, so it's important that it be clean, simple, and obvious. Unfortunately, solutions can end up long and complicated; such solutions rarely work, and in real life they would be brittle and hard to maintain. 30 points of your grade is based on how clear and easy to read your program is. My solution for `train.c` has 60 lines and my solution for `reaction.c` has 50 lines (not including comments). Note: your goal should be simplicity, not just line count; simple programs are

usually shorter than complex ones, but the shortest program isn't always the simplest.

Submitting the Lab

To submit your solutions, type the following:

```
make submit
```