PS 1 - Synchronization

Operating Systems, CS-UH 3010

Assigned:	Sept 20, 2022	Due:	Sept 27, 2022

Overview

Must be done individually!

A fancy Hyperloop system is being installed to transport Qatari and German fans from a far away parking lot into one of the new soccer stadiums for the FIFA world cup.

For safety reasons, the pods can only transport **four** fans at a time. Pods must always be full (four people) for transport.

In a pod, there can never be three fans for a given club and one fan from another club, otherwise they might overpower the minority fan and force them to switch teams.

You will write a procedure `arrives(f)`. The procedure is called by a Qatari or a German fan when he/she arrives at the hyperloop terminal. The fans are threads. `f` is a pointer to a structure holding information about each fan, such as its thread number, thread id, and club. The procedure arranges the arriving fans into safe pod loads; once the pod is full, only **one** thread calls `launch(...)`. Only after `launch(...)` completes can the threads `return`.

We provide you with a starting code base, with the following components:

1. **main()**, which reads an input string from stdio that contains the total number of fans and represents the order in which the fans arrive at the terminal.

   For example, the input string 8 1 1 1 1 0 0 0 0 indicates that 8 fans will arrive, four of them are Qatari fans (0) and four are German fans (1).

   For each fan, represented by `f`, a new thread is created and initialized with internal information regarding the club. Each thread is set up to call `arrives(f)`. Note that the order in which the threads execute is not guaranteed to match the order in which they arrive.

   After executing the threads, `main` waits for them to complete and then prints out for each pod, the fans within the pod, and the distribution of fans within it (`TWOS` or `FOURS`). Your code will hang if

2. **arrives(f)**, this is the procedure you need to setup correctly. For now it randomly sets up some threads as launchers and some threads as pod dwellers. It has no correct synchronization.
3. **launch(f, c)**, which `arrives(f)` calls. This procedure stores in shared buffers information on the thread calling launch and the pod distribution c which can either be `TWOS` or `FOURS`
4. **get_in_pod(f)**, which is called by threads that can go into pod but are not launching it. Each thread updates the shared buffers with their information.

We will evaluate your code with a separate testing module, which we did not share with you. Make sure you adequately test your code base.

You have to write two variants for this problem.

1. Semaphores-only implementation.
2. Condition-variables-only implementation.

You will have to write code to initialize additional, appropriate synchronization primitives and to write the code within `arrives(f)`.

Remarks

The starter code base uses POSIX threads. This problem set is a good way to get familiar with multi-threading and synchronization concepts. Mac OS X does not implement the POSIX `sem_init(...)`. This is why we use named semaphores and initialize them with the help of `sem_open(...)`. Named semaphores are stored permanently until physically destroyed. To make sure you don't introduce bugs from past runs of the code, make sure you destroy the semaphore before initialization with `sem_unlink()`.

As a result of your condition-variables only implementation, you can ***simplify*** the implementation of `launch(f, c)` and `get_in_pod(f)`. **Please feel free to change these modules only for the condition-variables implementation.**

You can compile and execute the code as follows:

```
gcc hyperloop.c -o hyperloop.o
echo "8 1 0 0 0 0 0 0 1" > test
./hyperloop.o < test
```

This could print the following:

```
4-German (fours),  8-German (pod),  3-German (pod),  1-German (pod)
7-Qatar (twos),  6-Qatar (pod),  5-Qatar (pod),  2-German (pod)
```

Your code should always exit if you have $4x$ German fans and $4y$ Qatar fans. Your code should hang if you have $4x + 3$ German fans and $4y + 1$ Qatar fans or $4x + 1$ German fans and $4y + 3$ Qatar fans.

Grading Rubric

This problem set is worth a total of 10 points.

	Deliverables	**Points**
1	Correct **semaphore** implementation that does not starve, deadlock or hang (except in appropriate situations), and solely uses sempahores for synchronization	4
2	Correct **condition variable** implementation that does not starve, deadlock or hang (except in appropriate situations) and solely uses condition variables for synchronization	3
3	Clean, well-documented, easy to understand code	2
4	Your solution has no busy-waiting and no starvation and you implement a testing module	1

Your solution should be starvation-free and there should be no busy waiting.

The correctness criteria for this project are:

1. Exactly four (no less, no more) fans are in a pod when it launches.
2. Either four fans from the same club or two from each club are in a pod when it launches.
3. Only one thread makes a call to launch for every four threads that end up in a pod.
4. Threads return only after the call to launch completes (we took care of this one for you for - look at the implementation of `launch` and `get_in_pod`).
5. If there are enough fans for a pod to launch, a pod will launch.