Due: Wednesday, Oct 15, 2014. 11:59pm (no extensions).

What to submit: A tar file containing files `water.c` and `watercond.c` containing your codes.

1. Practicing PThreads

To help get you started with project 2, this small exercise asks you to translate a concurrent program from pseudo code to code using the PThreads library.

In this example, which is taken from Downey's Little Book of Semaphores [1], available at http://greenteapress.com/semaphores/, a synchronization problem concerning two types of threads is considered. Some threads are "Oxygen O" threads whereas others are "Hydrogen H" threads. To form a molecule of water (H_2O), two hydrogen and one oxygen thread need to bond - that is, they have to agree that they're reached the point in their execution where they are ready to bond. This example isn't deep, but it demonstrates how to use semaphores, mutexes, and barriers to model a synchronization problem between multiple threads, along with proper protection of shared state.

The following code is reproduced, with minor changes, from Section 5.5 of Downey's book.

Listing 1: Variables

```
mutex = Semaphore(1)
oxygen = 0
hydrogen = 0
barrier = Barrier(3)
oxyQueue = Semaphore(0)
hydroQueue = Semaphore(0)
```

Listing 2: Oxygen code

```
mutex.wait()
oxygen += 1
if hydrogen >= 2:
    hydroQueue.signal(2)
    hydrogen -= 2
    oxyQueue.signal()
    oxygen -= 1
else:
    mutex.signal()

oxyQueue.wait()
bond()
```

```
barrier.wait()
mutex.signal()
```

Listing 3: Hydrogen code

```
mutex.wait()
hydrogen += 1
if hydrogen >= 2 and oxygen >= 1:
    hydroQueue.signal(2)
    hydrogen -= 2
    oxyQueue.signal()
    oxygen -= 1
else:
    mutex.signal()

hydroQueue.wait()

barrier.wait()
```

Note that hydrogen does not call bond(), unlike in Downey's code. In addition, you'll need a driver and a definition of bond():

Listing 4: Main Program

```
bonds = 0
function bond():
    printf("A H2O molecule is formed\n");
    bonds++;

function main():
    initialization
    ROUNDS = 100
    N = 30
    for k in 0..ROUNDS-1:
        for i in 0..N-1:
            if i % 3 == 0:
                spawn oxygen thread
            else
                spawn hydrogen thread

        for i in 0..N-1:
            join all spawned threads in this ROUND

        printf "%d H20\n", bonds
```

When run, your program should output:

```
A H2O molecule is formed
A H2O molecule is formed
A H2O molecule is formed
A H2O molecule is formed
A H2O molecule is formed
A H2O molecule is formed
A H2O molecule is formed
A H2O molecule is formed
A H2O molecule is formed
A H2O molecule is formed
10 H20
A H2O molecule is formed
......
A H2O molecule is formed
A H2O molecule is formed
1000 H20
```

Please use a `pthread_mutex_t` for `mutex`, a `sem_t` for `oxyQueue` and `hydroQueue`, and a `pthread_barrier_t` for `barrier`.

2. Practice Running Helgrind

Make sure your program runs quietly under the Helgrind race condition checker:

```
valgrind --tool=helgrind ./water
```

Helgrind will complain about Downey's idea of having the oxygen thread always release the mutex, even if it did not acquire it.

Change your code slightly such that always the thread (oxygen or hydrogen) who acquired the mutex releases it. Include `water.c` in your submission.

3. Practice Using Condition Variables

In part 3, replace the two semaphores with condition variables and adjust the logic accordingly. Your program should again run quietly under Helgrind.

Include `watercond.c` in your submission.

Bibliography

[1] Allen B. Downey. *The little book of semaphores*. http://greenteapress.com/semaphores/.