9sync-barrier-lab

CST 334: Operating Systems
Dr. Glenn Bruns

Lab: Synchronization barriers with locks and condition variables

A “synchronization barrier” is a synchronization concept in which no thread in a group can proceed until all threads in the group are ready to proceed.

In class we learned how to implement a synchronization barrier with semaphores. Here is the solution from “The Little Book of Semaphores”:

```
rendezvous

mutex.wait()
    count = count + 1
mutex.signal()

if count == n: barrier.signal()

barrier.wait()
barrier.signal()

critical point
```

In this lab we will implement a synchronization barrier with locks and condition variables. We will follow the “Anderson/Dahlin” method for writing shared objects. The file needed for the lab can be found on the hosting machine at:

`/home/CLASSES/brunsglenn/cst334/labs/sync-barrier/barrier-skeleton.c`

Copy this file to a file named `barrier.c` in a directory of your own, and make sure you can compile it:

```
$ gcc -pthread -lm -o barrier barrier.c
```

Unlike our last lab, not much code is supplied for you to start with. However, it’s not hard to implement a simple barrier. You may want to begin by creating a Makefile.

1. Implement the barrier code, and think about whether the existing printf statements are enough to convince you the code is right.

2. Did you use signal() or broadcast() in your solution? If you used signal(), create another version but this time using broadcast(). If you used broadcast(), create a version using signal().

3. If you still have time, time running performance tests on your two solutions.

4. If you still have time, try making your synchronization barrier reusable. This means that after the barrier is used, it is ready to be used again.

5. One approach to solving this problem is to try to translate the semaphore-based version above into a version using locks and condition variables. Is there a general method for translating code that uses semaphores into code that uses locks and condition variables? If you can think of a method, try to write a version of the synchronization barrier that is a translation of the semaphore-based code.

Cheat sheet: (this can also be found on our iLearn page)

To declare a lock:	`pthread_mutex_t lock;`
To initialize a lock:	`pthread_mutex_init(&lock, NULL);`
To acquire a lock:	`pthread_mutex_lock(&lock);`
To release a lock:	`pthread_mutex_unlock(&lock);`
To declare a condition variable:	`pthread_cond_t cv;`
To initialize a condition variable:	`pthread_cond_init(&cv, NULL);`
To wait on a condition variable:	`pthread_cond_wait(&cv, &lock);`
To signal a condition variable:	`pthread_cond_signal(&cv);`
To broadcast to a condition variable:	`pthread_cond_broadcast(&cv);`