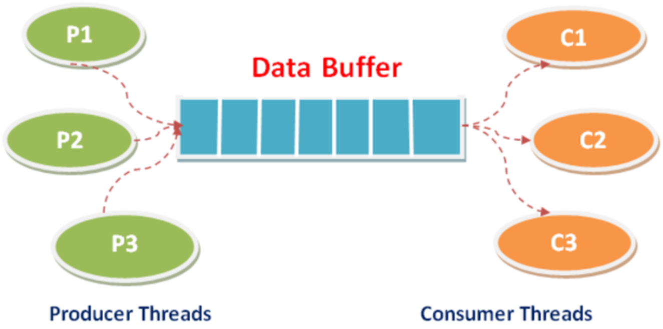
CS 360 Operating Systems **Winter 2019**
Assignment-3

Due Date: March 4th 2019

There are **two** problems in this assignment.

This is also a group assignment. You'll work with the same group that you worked in Assignment2. If you want to change your group, please let me know.

Problem-1: Producer-Consumer



In computing, the producer–consumer problem is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size data buffer/queue.

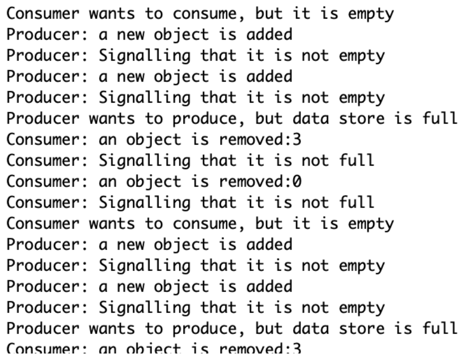
- The producer's job is to generate data, put it into the queue, and start again.
- At the same time, the consumer is consuming the data (i.e. removing it from the queue), one piece at a time.

Problem: Using locks and condition variable, we need to make sure that the producer won't try to add data into buffer if it's full and that the consumer won't try to remove data from an empty data buffer. You are given the implementations for the following Java files already:

- `App.java`
- `Producer.java`
- `Consumer.java`

You are asked to provide the producer-consumer implementation for `SharedDataStore.java` using locks and condition variables.

Sample Run



```
Consumer wants to consume, but it is empty
Producer: a new object is added
Producer: Signalling that it is not empty
Producer: a new object is added
Producer: Signalling that it is not empty
Producer wants to produce, but data store is full
Consumer: an object is removed:3
Consumer: Signalling that it is not full
Consumer: an object is removed:0
Consumer: Signalling that it is not full
Consumer wants to consume, but it is empty
Producer: a new object is added
Producer: Signalling that it is not empty
Producer: a new object is added
Producer: Signalling that it is not empty
Producer wants to produce, but data store is full
Consumer: an object is removed:3
```

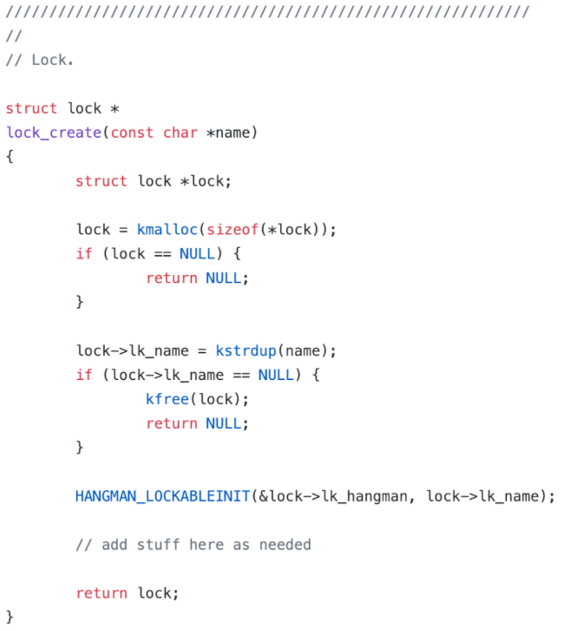
```java
//App.java
public class App {
      public static void main(String[] args) {
            SharedDataStore dataStore = new SharedDataStore(5);//max capacity
            Producer producer = new Producer(dataStore);
            Consumer consumer = new Consumer(dataStore);
            producer.start();
            consumer.start();
      }

}

//Producer.java
public class Producer extends Thread{
      private SharedDataStore dataStore;

      public Producer(SharedDataStore dataStore) {
            this.dataStore = dataStore;
      }

      @Override
      public void run() {
            while(true){
                  Random r = new Random();
                  int number = r.nextInt(10);
                  dataStore.produce(number);
            }
      }

}

//Consumer.java
public class Consumer extends Thread{
      private SharedDataStore dataStore;

      public Consumer(SharedDataStore dataStore) {
            this.dataStore = dataStore;
      }

      @Override
      public void run() {
            while(true){
                  dataStore.consume();
            }
      }

}
```

```
//SharedDataStore.java
public class SharedDataStore {

      //TODO: your code goes here



















}
```

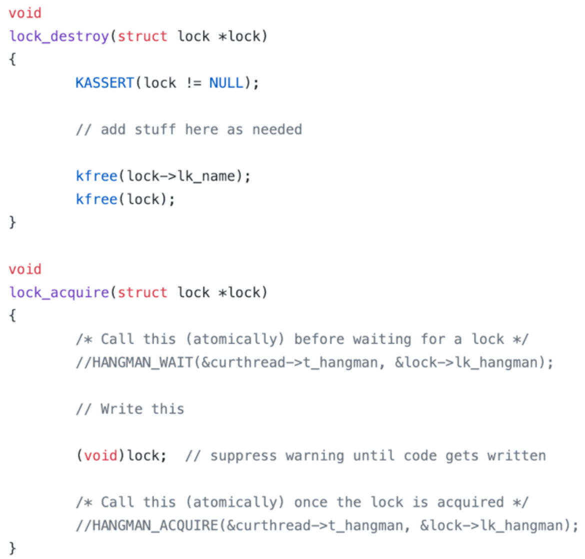
Problem 2: Implement Locks in os/161

OS/161 is an incomplete operating system. Locks are not implemented in OS/161. In this assignment, you are asked to implement the locks for OS/161. The interface for the lock structure is defined in kern/include/synch.h. Implementation for functions is provided in kern/threads/synch.c. You can use the implementation of semaphores as a model, but do not build your lock implementation on top of semaphores or you will be penalized. In other words, your lock implementation should not use sem_create(), P(), V() or any of the other functions from the semaphore interface. OS/161 also includes test cases for locks.

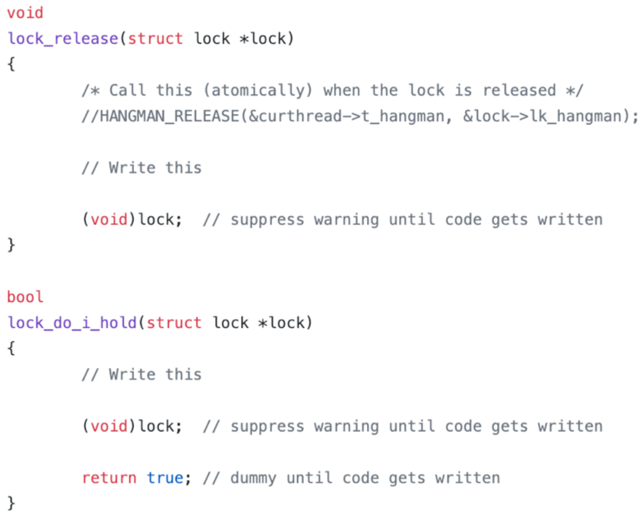
You are going to complete the implementation for the following lock functions. In class, while we were using pthread API, we called create, lock, unlock,... functions of thread API. In this assignment, you'll have the opportunity to write the logic behind these low level functions in OS/161. I hope you'll enjoy it!



```
////////////////////////////////////////////////////////////
//
// Lock.

struct lock *
lock_create(const char *name)
{
        struct lock *lock;

        lock = kmalloc(sizeof(*lock));
        if (lock == NULL) {
                return NULL;
        }

        lock->lk_name = kstrdup(name);
        if (lock->lk_name == NULL) {
                kfree(lock);
                return NULL;
        }

        HANGMAN_LOCKABLEINIT(&lock->lk_hangman, lock->lk_name);

        // add stuff here as needed

        return lock;
}
```

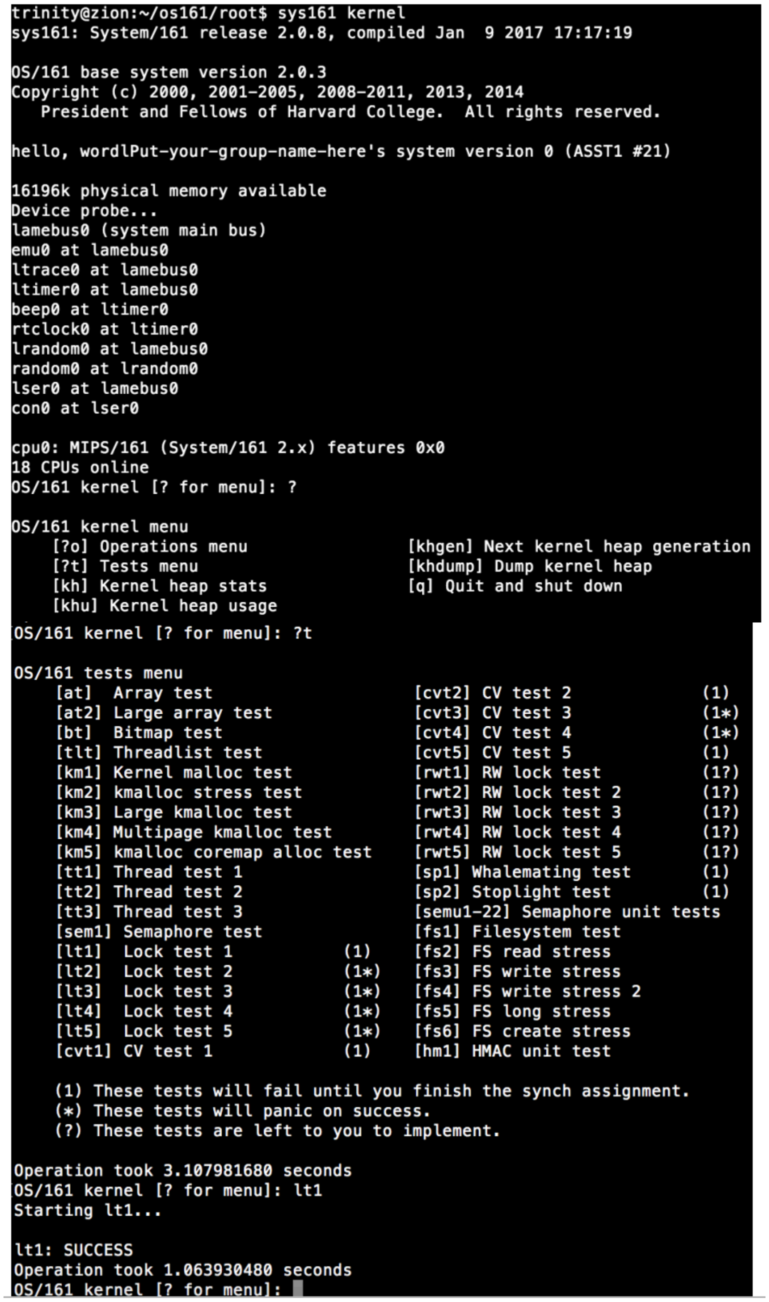


```
void
lock_destroy(struct lock *lock)
{
        KASSERT(lock != NULL);

        // add stuff here as needed

        kfree(lock->lk_name);
        kfree(lock);
}

void
lock_acquire(struct lock *lock)
{
        /* Call this (atomically) before waiting for a lock */
        //HANGMAN_WAIT(&curthread->t_hangman, &lock->lk_hangman);

        // Write this

        (void)lock;  // suppress warning until code gets written

        /* Call this (atomically) once the lock is acquired */
        //HANGMAN_ACQUIRE(&curthread->t_hangman, &lock->lk_hangman);
}
```



```
void
lock_release(struct lock *lock)
{
        /* Call this (atomically) when the lock is released */
        //HANGMAN_RELEASE(&curthread->t_hangman, &lock->lk_hangman);

        // Write this

        (void)lock;  // suppress warning until code gets written
}

bool
lock_do_i_hold(struct lock *lock)
{
        // Write this

        (void)lock;  // suppress warning until code gets written

        return true; // dummy until code gets written
}
```

In order to test you implementation, uou are required to run test, lt1 which provided by sys161. You should be receiving “Success” as the result of the test. A sample screenshot for running the test is given below.



```
trinity@zion:~/os161/root$ sys161 kernel
sys161: System/161 release 2.0.8, compiled Jan  9 2017 17:17:19

OS/161 base system version 2.0.3
Copyright (c) 2000, 2001-2005, 2008-2011, 2013, 2014
   President and Fellows of Harvard College.  All rights reserved.

hello, wordlPut-your-group-name-here's system version 0 (ASST1 #21)

16196k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrace0 at lamebus0
ltimer0 at lamebus0
beep0 at ltimer0
rtclock0 at ltimer0
lrandom0 at lamebus0
random0 at lrandom0
lser0 at lamebus0
con0 at lser0

cpu0: MIPS/161 (System/161 2.x) features 0x0
18 CPUs online
OS/161 kernel [? for menu]: ?

OS/161 kernel menu
    [?o] Operations menu                [khgen] Next kernel heap generation
    [?t] Tests menu                     [khdump] Dump kernel heap
    [kh] Kernel heap stats              [q] Quit and shut down
    [khu] Kernel heap usage
OS/161 kernel [? for menu]: ?t

OS/161 tests menu
    [at]  Array test                    [cvt2] CV test 2              (1)
    [at2] Large array test              [cvt3] CV test 3              (1*)
    [bt]  Bitmap test                   [cvt4] CV test 4              (1*)
    [tlt] Threadlist test               [cvt5] CV test 5              (1)
    [km1] Kernel malloc test            [rwt1] RW lock test           (1?)
    [km2] kmalloc stress test           [rwt2] RW lock test 2         (1?)
    [km3] Large kmalloc test            [rwt3] RW lock test 3         (1?)
    [km4] Multipage kmalloc test        [rwt4] RW lock test 4         (1?)
    [km5] kmalloc coremap alloc test    [rwt5] RW lock test 5         (1?)
    [tt1] Thread test 1                 [sp1] Whalemating test        (1)
    [tt2] Thread test 2                 [sp2] Stoplight test          (1)
    [tt3] Thread test 3                 [semu1-22] Semaphore unit tests
    [sem1] Semaphore test               [fs1] Filesystem test
    [lt1]  Lock test 1           (1)    [fs2] FS read stress
    [lt2]  Lock test 2           (1*)   [fs3] FS write stress
    [lt3]  Lock test 3           (1*)   [fs4] FS write stress 2
    [lt4]  Lock test 4           (1*)   [fs5] FS long stress
    [lt5]  Lock test 5           (1*)   [fs6] FS create stress
    [cvt1] CV test 1             (1)    [hm1] HMAC unit test

    (1) These tests will fail until you finish the synch assignment.
    (*) These tests will panic on success.
    (?) These tests are left to you to implement.

Operation took 3.107981680 seconds
OS/161 kernel [? for menu]: lt1
Starting lt1...

lt1: SUCCESS
Operation took 1.063930480 seconds
OS/161 kernel [? for menu]:
```

What to Submit:

- You are asked to submit your work as a single zip file via CANVAS. Zip file will include two folders
    - Problem1: includes your work as an eclipse Java project. Please don't forget to put comments in your source code explaining your reasoning for the solution.
    - Problem2: includes the kernel code that you updated (only the source files that you updated), and a document with an explanation for your implementation and with sample screenshots showing success from test-runs using OS/161.