CST 334: Operating Systems

Dr. Glenn Bruns

# Lab: Fully Synchronized Buffer

In this lab we will continue work on our synchronized buffer code by turning the synchronized buffer object into a “fully-synchronized” buffer object. Please work with a partner or partners.

A fully-synchronized buffer is like a synchronized buffer, except that a thread will not return from its write() call until the written value has been read. In other words, a write operation should do the following::

* wait until the buffer is empty
* write to the buffer
* wait until the written value is read

Use this code as your starting point:

/home/CLASSES/brunsglenn/cst334/labs/cond-vars-dahlin/fully\_sync\_buf\_skeleton.c

1. Modify the code to implement a fully-synchronized buffer.

Rename your code to fully\_sync\_buf.c There are some comments in the code containing YOUR CODE HERE. Replace these comments with zero or more lines of your own code.

1. Here are some things to things to think about:
   * what condition will you check to see if the written value has been read?
   * do you want/need to add another condition variable?
   * remember that when you use a condition variable you will always perform the wait operation within a loop.
2. If you think your code is working, do a test to make sure. For example, you might want to add print statements, and to use awk. If you can't get your code to work, see 'solution outline' at the bottom of the hints.
3. Have you considered this issue: is it possible, in your code, for a thread to be waiting for fsbuf\_write() operation to complete, and in the meantime other threads are performing their write operations?
4. If you believe your code is working, modify it so that it creates three readers and three writers. Does your code still work?
5. If you still have have, modify your code so that it supports a buffer of any size (but a fixed size).

## 

## Hints:

1. -
2. It is not necessary to add new condition variables. Is your code deadlocking? Do you need to add a signal call to the fsbuf\_
3. You may want to put print statements just before and after where the actual writing is done in fsbuf\_write(), like this:

printf("write starts\n");

fsbuf->count = 1;

fsbuf->buf = val;

printf("write ends\n");

and similarly for when actual reading is done. You may also want to print some kind of 'start' and 'end' message before and after the code where the fsbuf\_write() operation is waiting for the written value to be read by another thread.

Solution outline: