CS 4375 Spring 2020 OS Lab 5 – Producer-Consumer

This lab is inspired by a real-world scenario from NASA. We have a spacecraft that controls two different antennas. These antennas use data from receivers to generate control signals. The receivers come in pairs, in case one fails. We refer to the pair as Prime and Redundant. In total, we have 4 receivers, but we only need two signals, and we only care about the most recent data. The spacecraft antenna controller needs the signal from both antennas at the same time.

For our problem, we will model this as a producer-consumer. We have two channels, A and B, that we need to read. Each channel has a prime and a redundant producer. For our purposes, we will call these P and R. Therefore, we have AP, AR, BP, and BR. AP and BP produce a signal every 1.00 seconds. AR and BR produce a signal every 1.15 seconds. AP and AR write to one buffer, and BP and BR write to another buffer. Each buffer holds one data point. If the buffer is empty, the producer fills it. If the buffer has data when the prime producer generates a new value, the prime producer overwrites the data in the buffer. If the buffer has data when the redundant producer generates a value, the redundant's data is thrown away.

For example, The following table shows the prime and redundant generating values at their respective rates. The buffer indicates the value contained in the shared buffer. The consumer may read at any time.

time	P	R	buffer	
1.0	1	-	1	
1.25	-	1.1	1	
1.50	-	-	1	
1.75	-	-	-	read by consumer
2.00	2	-	2	
2.25	-	-	2	
2.50	-	2.1	2	
2.75	-	-	-	read by consumer
3.00	3	-	3	
3.25	-	-	3	
3.50	-	-	-	read by consumer
3.75	-	3.1	3.1	
4.0	4	-	4	
4.25	-	-	4	

The producers will use the following pseudo-code:

```
do_forever {
    wait delay_time;
    data = gen_new_data();
    write_data (data, isPrime);
}
```

Here, the writes are non-blocking. The Prime producer can always overwrite a buffer. There is an example of producer code in the producer.c file. You will need to make some modifications to meet the requirements of this lab, namely, you will need 4 threads, one for each producer.

The consumer will use the following pseudo-code:

```
do forever {
    a_val = read(a_buffer);
    b_val= read(b_buffer);
    printf(<time stamp> , a_val, b_val);
}
```

Here, the reads are blocking reads. The consumer will acquire data from channel A and channel B. As soon as it gets data from both channels, it prints. The time stamp needs to print the seconds from the start of execution to the nearest millisecond (3 digits to the right of the decimal point, “%6.3f”). The data values should be printed to the nearest hundredth, (“3.2f”). The format should be

“%6.3f, %3.2f, %3.2f\n”

Note the commas and end of line.

To complete this assignment correctly, you will start 5 threads. You must use two buffers, each of which has one entry. These buffers must be protected from update collisions by locking and unlocking. You must implement a blocking read for the consumer. You’ll need to modify the standard producer-consumer code so that writes are non-blocking.

```
#include <stdio.h>
#include <time.h>
#include <math.h>
#define TRUE 1
#include <sys/time.h>

/* Sample code for a producer, except it doesn't run in a thread.
 * In this code, producer takes three arguments: an int (0 for
 * false, 1 for true) indicating whether the producer is prime;
 * and two longs representing the number of seconds and nanoseconds
 * to delay between productions. The output is in the form
 *    <nsec since start>, <sim data>
 * Timing is approximate. It does not take into account processing
 * time.
 *
 * Note: must be linked with -lm    (e.g., gcc -lm ...)
*/

struct timeval start;

int main (char *argv[], int argc) {
  producer(1, 2L, 500000000L);
}

int producer (int isPrime, long secDelay, long nsecDelay) {
   char data[80];
   gettimeofday (&start,0);
   while (TRUE) {
       struct timespec tspec1, tspec2;
       tspec1.tv_sec = secDelay;
       tspec1.tv_nsec = nsecDelay;
       nanosleep(&tspec1, &tspec2);
       gen_new_data(&data);
            write_data (data, isPrime);
            }
}

int gen_new_data (char *out) {
   struct timeval now;
   gettimeofday (&now,0);
   double delta = (now.tv_sec - start.tv_sec)
            + (double)(now.tv_usec - start.tv_usec)/1000000L;
   double data = sin(delta);
   sprintf(out, "%6.3f, %3.2f", delta, data);
}

int write_data (char *data, int isPrime) {
   printf("%s\n", data);
}
```