CSC 139, Spring 2015
Department of Computer Science, CSUS
3/4/2015

Programming Assignment 3

- **The due day: March 19.**
- **No credit for a program that does not compile or does not run.**

In this assignment, you are to write a multi-threaded C program for a *bounded-buffer producer/consumer* application: file copying. You are required to use **semaphores** in your program to enforce mutually exclusive access to the buffer and to synchronize the producer and the consumer with regard to buffer slots/items. Make sure that your submitted program compiles and runs on Athena, the ECS Linux machine.

1. Your program should be run with two file names provided at the command line, say: **infile** and **outfile**. Otherwise, your program prompts the user with the message: "Correct Usage: prog3 infile outfile" and then terminates.
2. In your program, the main thread verifies the input, opens the **infile**, and creates the **outfile**. The main thread then spawns a *producer* thread and a *consumer* thread, and waits for both the producer and the consumer to finish before it terminates. The producer and the consumer share **a buffer of 9 slots with each slot having a size of 18 bytes**. The producer reads a string of 18-byte at a time from the **infile** and places it into the next available buffer slot. The consumer takes the next available string from a buffer slot and writes it into the **outfile**. **The outfile thus is a verbatim copy of the infile**.
3. The buffer can only be accessed in a *mutually exclusive* fashion, which is enforced through the use of a semaphore variable **`buf_lock`** and the **sem_wait** and **sem_post** operations.
4. When the buffer is *full*, the producer must wait until a buffer slot becomes free before it can place a string into it. When the buffer is *empty*, the consumer must wait for a string to be available. When the producer fills a slot, the consumer is notified of an item available; when the consumer empties a slot, the producer is notified of a slot available. These synchronization conditions between the producer and the consumer are facilitated by two additional semaphore variables **`slot_avail`** and **`item_avail`,** and the **sem_wait** and **sem_post** operations.
5. Because the order in which the semaphore wait operations (**sem_wait**) are executed can be a cause for *deadlock*, care must be exercised when using **sem_wait**. From the producer's perspective, you want to check to see if there is a buffer slot available (through **`slot_avail`**) before attempting to gain exclusive access to the buffer (via **`buf_lock`**). Similarly, from the consumer's standpoint, you need to make sure that there is at least an item (a string) in a buffer slot (via **`item_avail`**) before the consumer attempts to have exclusive control of the buffer (via **`buf_lock`**).
6. To be able to use semaphores in your program, include <**semaphore.h**>. Use **sem_t** to define **`buf_lock`**, **`slot_avail`**, and **`item_avail`**. Then use **sem_init** to initialize these variables:

   ```
   sem_init(&buf_lock, 0, 1);
   sem_init(&slot_avail, 0, 9);
   sem_init(&item_avail, 0, 0);
   ```

   If *x* is a semaphore variable, then we can have **sem_wait**(&*x*) and **sem_post**(&*x*). When compiling your program in a Linux machine, use an additional switch **"–Lposix4**" (For a Solaris machine, the switch is **"–lposix4**").
7. The data type for the buffer are given as follows:

   ```
   #define  SLOTSIZE  18
   #define  SLOTCNT   9
   char     buffer[SLOTCNT][SLOTSIZE];
   ```

You may need to have additional variables to handle things such as (a) the *next available slot* for the producer; (b) the *next available item* for the consumer; (c) *number of items available* in the buffer; (d) *number of bytes* in a slot; and (e) a *flag* to indicate when the producing/consuming process ends.

8. Test your program with your own data. Then make sure that your program works for the **infile** that contains the following:

```
Semaphores are effective tools for critical section solution and
synchronization of concurrent activities. In this exercise, we will
use semaphores to implement the bounded-buffer producer/consumer
approach for a file copying application. Specifically, you need to
create a producer thread and a consumer thread that work concurrently
with a buffer of 9 slots. The producer produces (reads) a string of
18-byte at a time from the input file, i.e., this file, and places
(writes) the string to the next available slot in the buffer in a
mutually exclusive manner. On the other hand, the consumer fetches a
an item (18-byte string) from the next filled buffer slot at a time,
and consumes it (writes it to the output file). When the process is
finished, the output file is a verbatim copy of the input file.
Three semaphores are used to enforce the mutually exclusive access to
the buffer and to synchronize the activities of the producer and
the consumer. A final note: the last string processed may not be
exactly 18-byte long.
```

9. Submission Requirements. Your program must include adequate commenting (**points deduction for programs with inadequate comments**). Compile your program using the following:

```
gcc   prog3.c  -o  prog3  -lpthread  -Lposix4
```

Test your program to make sure it works correctly for the following scenario.

```
prog3  infile
ls
cat  infile
prog3  infile  outfile
cat  outfile
```

Save your program as **`yourName_prog3.c`** and submit it to “Program 3 Submission” in SacCT.