Assignment 3 – Producer-Consumer with Semaphores

Assigned: Tuesday, March 8, 2011
Due: Tuesday, April 5, 2011, 1pm (as instructed below)

Assignment Description

You are to complete the "Producer-Consumer Programming Project" which is described in the required textbook *Operating System Concepts (8th Edition)* by Silberschatz et al. on pages 274-280. The project asks you to implement a semaphore-based solution to the producer-consumer problem by using the pthread library.

Important Notes

The followings are requirements in addition or in replacement of what is stated in the description in the textbook. These additional requirements might necessitate you to change some of the code snippets given in the textbook.

- Though the project description in the textbook provides Win32 API in addition to the pthread API, you are not allowed to use the Win32 API to develop your code.
- Your code will be tested and graded on the UNIX machines at the ECC Lab.
- Name your executable as "`prodcon`"
- Each producer and consumer should "know" its unique ID as given by the main process. This will ease implementation of the following requirements.
- The producers must report production of an item by printing out a message in this format:
  ```
  Producer <ID> produced <item ID>
  ```
  where <ID> is the ID of the producer and <item ID> is the random ID of the item generated randomly as described in the textbook.
- The consumers must report consumption of an item by printing out a message in this format
  ```
        Consumer <ID> consumed <item ID>
  ```
  where <ID> is the ID of the consumer and <item ID> is the random ID of the item generated randomly as described in the textbook. Notice that there must be one level of indentation preceding a consumer's output.
- When the sleep time is over, the main process must wake up and print out the number of items produced and consumed in the following format:
  ```
  Items produced: <number of produced items>
  Items consumed: <number of consumed items>
  ```
- The format of your program's output should look like as follows:
  ```
  Producer 1 produced 34139
  Producer 2 produced 9893424
        Consumer 1 consumed 34139
  Producer 3 produced 465923
        Consumer 3 consumed 9893424
  ...
  ...
  ...
  Items produced: 12
  Items consumed: 10
  ```

Graduate Students (extra credit for others)

The followings are further requirements for graduate students. The undergraduate students will obtain extra credit if they satisfy these further requirements.

- In order to improve the performance of `prodcon` solving the Producer/Consumer problem, we would like to add an additional buffer so that producers and consumers do not have to wait for each other when writing to a common buffer. In order to leverage this additional buffer, we introduce a new type of thread into the system, called *Mover*. The job of the movers is to take produced items from the buffer designated for producers and put them into the buffer designated for consumers. Implement this new type of thread in your code to solve this Producer/Mover/Consumer problem.

- Name your new executable as "`prodmovcon`"
- `prodcon` was receiving three parameters from the command line. For `prodmovcon`, add one more fourth parameter describing the number of mover threads. We will run your `prodmovcon` as follows, if we would like to have 5sec of sleep time, 10 producers, 11 consumers and 12 movers:

  "`$prompt> ./prodmovcon 5 10 11 12`"
- The buffer in `prodcon` was hardcoded to be of size 5. Make the size of the additional buffer in `prodmovcon` as 10, but keep the original buffer at size 5.
- Each mover should "know" its unique ID as given by the main process. This will ease implementation of the following requirements.
- The producers must report production of an item by printing out a message in this format:
  ```
  Producer <ID> produced <item ID>
  ```
  where <ID> is the ID of the producer and <item ID> is the random ID of the item generated randomly as described in the textbook.
- The movers must report movement of an item by printing out a message in this format
  ```
        Mover <ID> moved <item ID>
  ```
  where <ID> is the ID of the mover and <item ID> is the random ID of the item generated randomly as described in the textbook. Notice that there must be one level of indentation preceding a mover's output.
- The consumers must report consumption of an item by printing out a message in this format
  ```
              Consumer <ID> consumed <item ID>
  ```
  where <ID> is the ID of the consumer and <item ID> is the random ID of the item generated randomly as described in the textbook. Notice that there must be two levels of indentation preceding a consumer's output.
- When the sleep time is over, the main process must wake up and print out the number of items produced and consumed in the following format:
  ```
  Items produced: <number of produced items>
  Items moved: <number of moved items>
  Items consumed: <number of consumed items>
  ```
- The format of your program's output should look like as follows:
  ```
  Producer 1 produced 34139
  Producer 2 produced 9893424
        Mover 1 moved 34139
  Producer 3 produced 465923
              Consumer 1 consumed 34139
        Mover 4 moved 9893424
              Consumer 3 consumed 9893424
  ...
  ...
  ...
  Items produced: 12
  Items moved: 10
  Items consumed: 8
  ```

Submission Requirements

A **`makefile`** is required. All files in your submission will be copied to the same directory, therefore do not include any paths in your **`makefile`**. The **`makefile`** should include all dependencies that build your program. If a library is included, your **`makefile`** should also build the library.

To make this clear: ***do not hand in any binary or object code files***. All that is required is your source code, a **`makefile`**, and other necessary files as stated in the assignment description. Do test your project by copying the source code only into an empty directory and then compile it by entering the command **`make`**.

Your code *must* compile without any errors or warnings and run properly under the Linux system used in the ECC lab (check their Web site for hours). You may develop and test your programs on your own UNIX machine, but it is your responsibility to ensure that they also work properly on the Linux installation of ECC, under the default ECC Lab shell. There will be a heavy penalty if they don't. The specific programming language you use is your choice, but your **`makefile`** has to be compiling your source code and producing a program that is executable in the Linux installation of ECC.

Your source code will be assessed and marked. Commenting is definitely required in your source code.

Go through the following steps to make your submission file ready to submit:

1. Go to your home directory and make a folder named with your name in format LastName_FirstName.
2. Put your source code and all other necessary files in this folder
3. In your home directory type and enter:
   ```
   tar -cvf <LastName_FirstName>.tar <LastName_FirstName>
   ```
4. Finally, type and enter:
   ```
   gzip <LastName_FirstName>.tar
   ```

These above steps will yield a new file named <LastName_FirstName>**`.tar.gz`** in your home directory. To submit your file, log in to WebCampus and select the CS 446/646 course. From the "Course Tools" panel on left go to 'Assignments" and choose "Programming Assignment 3". Attach your submission file and submit.

If you have any problems please e-mail the instructor or the TA. ***Submissions including multiple files or sent by e-mail will NOT be evaluated.*** Do not turn in printouts in any form. Late assignments will be marked down according to the late policy published on the course Web page.

Good luck!