EECS 111 - System Software

Spring 2010

Assignment 3

Posted: Tuesday, April 27, 2010

Due: Tuesday, May 4, 2010, 12pm (noon)

A. Discussion: Process Synchronization

The goal of these exercises is to review and clarify the understanding of essential aspects covered in recent lectures.

- Race conditions: Exercise 6.8, and 6.15
- Synchronization primitives: Exercises 6.11, 6.12, 6.13, 6.14, and 6.18
- Critical sections: Exercise 6.16 and 6.22

These topics will be discussed at the beginning of this week's discussion session.

B. Project: Producer Consumer Problem, Threads, Busy Waiting

The goal of this project assignment is to implement the classic producer consumer problem by use of threads in order to practice thread creation and termination using the pthread API. At the same time, we will investigate inter-thread communication via a bounded buffer and synchronization using busy waiting.

As before, we will use the EECS department servers which offer parallel execution of threads on multiple CPUs.

Step 1: Setup

- We will use the very same setup as in the previous project. Please refer to the instructions of Assignment 1 and 2 for details.
- For this project, we will write a producer consumer application which will be reused for the coming Assignment 4. To get started, create a new source code file and name it **`prodcons.c`**. Since we will reuse the Fibonacci computation as worker load, you may want to copy your **`fibonacci(n)`** function from Assignment 2 into **`prodcons.c`**.

Step 2: Create a producer and a consumer thread

Your program should create two new threads (running independently from the main thread) that execute the functions **`produce()`** and **`consume()`** in order to act as producer and consumer, respectively. The producer will compute a sequence of Fibonacci numbers (using the **`fibonacci()`** function from the previous assignment) and send each computed number to the consumer via a **`send()`** function.

The consumer, in turn, will receive the numbers via a **`receive()`** function and check whether or not the sequence of numbers received matches the Fibonacci series correctly. To check the correctness, the consumer will compare each received number to the sum of the previous two received numbers. To keep it simple, the consumer does not need to check the initial two numbers (he just receives them).

Your program should contain the following functions:

```
int fibonacci(int n);
void *produce(void *arg);
void *consume(void *arg);
void send(int n);
int receive(void);
int main(int argc, char **argv);
```

Similar as before, your program should take an integer **`max`** from the command line. This value should be passed to the consumer and producer threads as argument **`arg`**. Both consumer and producer should work on the Fibonacci series from **`0`** to **`max`** and then terminate. When both threads have completed their execution, the main thread should exit the program.

Step 3: Bounded buffer with busy-waiting synchronization

For communication between the producer and consumer, we will use the bounded buffer implementation discussed in the text book in Section 3.4.1. This is the same as the bounded buffer version 1 discussed in Lecture 9. However, for this project, we will use simple integer values (type **`int`**) as items in the buffer and implement the buffer in separate **`send()`** and **`receive()`** functions called by the producer and consumer, respectively.

Note that we will improve this communication in the next Assignment 4.

Step 4: Compile and test your implementation

Instrument your program with suitable reporting functions (using **`printf()`**) so that you can observe the progress of the main, consumer, and producer threads and their communication.

Hint: Indent the lines printed by the consumer so that it is easier to distinguish it from the producer!

When running the program, your execution log should look similar to this:

```
malibu.eecs.uci.edu % ./prodcons 5
Creating producer of Fibonacci numbers...
Creating consumer of Fibonacci numbers...
Producer: computing fibo(0)...
Producer: sending number '0'...
                        Consumer: receiving fibo(0)...
                        Consumer: receiving fibo(1)...
Producer: computing fibo(1)...
Producer: sending number '1'...
Producer: computing fibo(2)...
Producer: sending number '1'...
Producer: computing fibo(3)...
Producer: sending number '2'...
Producer: computing fibo(4)...
Producer: sending number '3'...
Producer: computing fibo(5)...
Producer: sending number '5'...
Producer: max reached, exiting...
                        Consumer: receiving fibo(2)...
                        Consumer received '1'...Correct!
                        Consumer: receiving fibo(3)...
                        Consumer received '2'...Correct!
                        Consumer: receiving fibo(4)...
                        Consumer received '3'...Correct!
                        Consumer: receiving fibo(5)...
                        Consumer received '5'...Correct!
                        Consumer: max reached, exiting...
Done.
malibu.eecs.uci.edu %
```

We will again analyze this program using the **`/usr/bin/time`** command to measure the user, system, and elapsed execution times. Specifically, we will use **`max=42`** to compute and check the Fibonacci series up to the 42nd number.

State your expectation of the execution times. Then, list your measured times in a table (note the server name you are using) and briefly explain how this matches your expectation.

The relation of user and elapsed time is often referred to as *CPU load*:

$$\text{Load} = t_{user} / t_{elapsed}$$

Compute the CPU load for this example as a percentage and briefly explain what it means.

Finally, we know that the EECS department servers provide multiple CPUs. Your program should run fine on them. However, would it run fine as well on a single-CPU machine? Why?

Deliverables:

1. Statement: “I have read the Section on Academic Honesty in the UCI Catalogue of Classes (available online at http://www.editor.uci.edu/catalogue/appx/appx.2.htm#gen0) and submit this work accordingly.”
2. Source file **`prodcons.c`**, execution log **`prodcons.log`**, and a brief text file **`prodcons.txt`** explaining the results and answering the above questions (3-4 paragraphs) [100 points].

Submission instructions:

To submit your homework, send the deliverables in an email with subject “EECS111 HW3” to the course instructor at doemer@uci.edu.

To ensure proper credit, be sure to send your email before the deadline: Tuesday, May 4, 2010, at 12:00pm (noon).

--

Rainer Doemer (EH 3217, x4-9007, doemer@uci.edu)