Lab 4 – Process Synchronization

Due: 9:55 AM, Thursday, March 11, 2005

This week's lab consists of questions to look at on your own (not to be turned in) and some questions to be turned in. The programming for this lab is considered a "laboratory program" for honor code purposes. You may discuss the program with others, but what you turn in must be your own work.

Practice Questions

You do not need to turn in answers to these questions. But they could certainly form the basis for potential exam questions.

- SG&G 6.4, 6.9, 6.15
- Does Peterson's Algorithm work when scheduling is non-preemptive?

Lab Questions

Answer these questions in a plain text file named `lab4.txt` or in a PDF file named `lab4.pdf`.

1. SG&G 6.6 (2 points)
2. Consider the Bakery Algorithm from class. Explain why the following is true:

   If P_i is in its critical section, and P_k ($k \neq i$) has already chosen `number[k]` $\neq 0$, then `(number[i],i) < number[k],k)`.

   This does not need to be a formal proof, just a convincing explanation. (3 points)
3. *The Dining Savages.* A tribe of savages eats communal dinners from a large pot that can hold M servings of stewed missionary. When a savage wants to eat, he helps himself from the pot, unless it is empty. If the pot is empty, the savage wakes up the cook and then waits until the cook has refilled the pot. The behavior of the savages and cook is defined by the following processes:

   $Savage_i$: while (1) { get serving from pot; eat; }
   Cook: while (1) { sleep; put M servings in pot; }

   Develop psuedocode (along the lines of the in-class Sleeping Barber solution) for the actions of the savages and the cook. Use semaphores for synchronization. The only operations permitted on the semaphores are initialization, wait, and signal. You may assume the semaphores are fair. Your solution should avoid deadlock and awaken the cook only when the pot is empty. You do not need to implement C code for this part, just a list of shared variables and pseudocode describing the actions of the savages and the cook. (10 points)

Bounded Buffer with Semaphores

Write a C program that implements the bounded buffer problem for an arbitrary number of producers and consumers (specified by command-line parameters). Use POSIX threads to create your producers and consumers and use POSIX semaphores for synchronization. Your solution should avoid the busy wait from the class examples (other than any busy waiting that takes place inside a semaphore wait or signal operation).

Write your program in a file called `prodcons.c`. You may use the source code from any of the class examples as your starting point. (5 points)

Submission and Evaluation

- All necessary files should be submitted using `turnin` as a single "tar" file, `lab4.tar`. Include a `Makefile` to allow easy compilation of the C program.