Assignment 3: Semaphores

Recitation for ECES 338
February 15, 1999

I. Introductory Stuff

A. Assignment #1 has been graded – see your T.A.

B. Clean-up Machines – When coding with multiple processes, it is **imperative** that you clean-up after yourself.

C. Midterm will focus on Concurrent Programming (Chapter 6 from the Dinosaur text).
   1. I'll try and be less vague in the next recitation…

II. Semaphores

D. This explained fairly well in the Dinosaur text, but in essence, a *semaphore* is essentially a variable that can be operated on atomically, to ensure that processes have sole access to a specific set of code.

E. There are two operations intrinsic to *semaphores*:
   1. Wait (P) – blocks until the *semaphore* is zero, and then decrements the *semaphore* by one.
   2. Signal (V) – increments the *semaphore* by one.

F. From these two constructs, it is possible to construct some very elaborate concurrency-control hierarchies, as the text demonstrates.

III. Deadlock and Starvation (6.4.3)

G. There is a whole chapter (Chapter 7) in the Dinosaur text about what a *deadlock* is, and how to avoid it.
   1. Basically, a *deadlock* occurs when you have at least two processes that cannot continue, because they are both waiting for the other process to accomplish some task.
      (i) There is a very lucid example of *deadlock* in the Dinosaur text, on page 171.
   2. I'm not sure how the grader is going to deal with this in your assignments. I would tell you to work very hard to eliminate any possibility of *deadlock* in the algorithms that you submit.
      (ii) Try simulating a run through your algorithm (on paper), varying the order in which processes are scheduled to run, etc.

H. As to starvation, I'm not so sure that you have to worry about this for Assignment #3.
   1. If I change my mind, I'll post this information to my website.

IV. Shared Memory

I. Without going into much detail, shared memory allows you to declare variables that can be *"shared"* among several processes.
   1. Thus, it would be possible to update a counter variable in one process, and allow another process to have instant access to that information.

J. For the purposes of this assignment, you should explicitly state which of your variables are to be shared among which processes.

V. Hints 'n Kinks for Assignment #3:

K. Problem #1:
1. Although it is ambiguous, I think that you should assume that your answer should work for *multiple* car processes. If this changes, I will post it to my website.
2. Furthermore, I believe that it's fully possible to implement this problem using only four semaphores and no shared variables. However, you are allowed to use as many semaphores as you need, as well as shared variables.
3. The best way to attack this problem (or all of them, for that matter), is to break down each process into the activities that it must perform, and think about how the activities can be performed in a concurrent environment.

L. Problem #2:
1. I'm still a little confused by this problem. Basically, it looks like you **will** need to use shared variables for this question.
2. The thing to recognize is that an *A-type* process should keep a count of the number of *B-type* processes that it has seen. This means that it is possible for two *B-type* processes to be in the room at separate times, but still cause an *A-type* process to exit.
3. As I get questions about this problem, I will post further information to my website.

M. Problem #3:
1. I had to implement nearly the exact same problem in ECES 423.
   (i) The only difference for me was that I had to use threads and a monitor.
2. While you don't have to worry about threads, you might want to consider using a monitor for this problem.
   (ii) I haven't thought-through this fully, but it may make your algorithm simpler.
   (iii) Basically, you break-down each operation, into parts that can be done concurrently on the list, and parts that need mutual exclusive access.