Fall 2010
CSC 4304 – Systems Programming
Homework Assignment #3

The due date is: November 4th, Thursday, before the class. Late submission is **not** allowed.

Problem 1: Consider a process with two concurrent threads T1 and T2. The code being executed by T1 and T2 is as follows:

Shared Data:
X:= 5; Y:=10;

T1:	**T2:**
Y = X+1; X = Y; Write X;	U = Y-1; Y = U; Write Y;

(a) Assume that each assignment statement on its own is executed as an atomic operation. What are the possible outputs of this process? Please show your work.

(b) Where would be the correct place to put semaphores to prevent inconsistencies but also allow some level of parallelism? Please show how you would use semaphores to achieve this. (For this one, assume that the order in which X and Y are written is not important as long as the have the correct values.)

Problem 2: Imagine you are writing the code to manage a hash table that will be shared among several concurrent threads. Assume that operations on the table need to be atomic. You could use a single mutual exclusion lock to protect the entire table, or you could devise a scheme with one lock per hash-table bucket. Which approach is likely to work better, under what circumstances? Why?

Problem 3: Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system? How?

Problem 4: The dining philosophers problem is a classic exercise in synchronization. Five philosophers sit around a circular table. In the center is a large communal plate of spaghetti. Each philosopher repeatedly thinks for a while and then eats for a while, at intervals of his or her own choosing. On the table between each pair of adjacent philosophers is a single fork. To eat, a philosopher requires both adjacent forks: the one on the left and the one on the right. Because they share a fork, adjacent philosophers cannot eat simultaneously.

Write a solution to the dining philosophers problem in which each philosopher is represented by a POSIX thread and the forks are represented by shared data. Synchronize access to the forks using semaphores. Try to maximize concurrency.