Homework 2

Due Date : April 29, 1999 **Points** : 100

Short-Answer Questions

These can be answered in a sentence or two, and are intended to reinforce important points.

1. (*3 points*) What is a race condition? (Tanenbaum, problem 2.5)
2. (*4 points*) What are the Bernstein conditions?
3. (*4 points*) What conditions must a solution to the critical section problem meet?.
4. (*4 points*) Suppose someone devised a new synchronization primitive she claimed were as powerful as semaphores. How would she prove this claim?

Long-Answer Questions

These questions require some thought and longer answers than the short-answer questions. They are intended to have you use the concepts discussed in class, to be sure you understand them and can work with them.

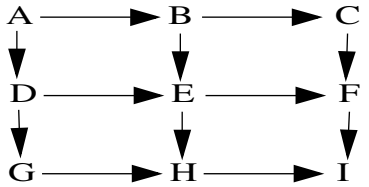
5. (*15 points*) A *binary semaphore* is most commonly defined as a semaphore whose integer value can range only between 0 and 1. (This is slightly different than Tanenbaum's definition on p. 68, but is more common.) Implement counting semaphores using binary semaphores. (variation of Tanenbaum, problem 2.11)
6. (*20 points*) Instead of test-and-set, some computers provide an atomic instruction that sets the new value to 1 greater than its old value, as shown here:

```
function testandinc(var lock: integer): integer;
begin
      testandinc := lock;
      lock := lock + 1;
end (* testandinc *)
```

   Show how to use this instruction to implement locks. (Hint: you must be wary of integer overflow.)
7. (*25 points*) On page 77 of the text, Tanenbaum claims that the program in Fig. 2-18 solves the Dining Philosophers' problem. Either prove or disprove this claim. If the claim is false, fix the program (that is, implement a solution to the dining philosophers' problem using semaphores).
8. (*25 points*) Synchronization within monitors uses condition variables and two special operations, *wait* and *signal*. A more general form of synchronization would be to have a single primitive, *waituntil*, that had an arbitrary Boolean predicate as parameter. Thus, one could say, for example,

```
waituntil x < 0 or y + z < n
```

   The *signal* primitive would be no longer needed. This looks more general than that of Hoare or Brinch Hansen, but is not used.
   a. Show that *waituntil* is as general as Hoare's scheme (that is, any monitor using Hoare's *wait* and *signal* primitives can be written using it.) Is it in fact more general? Justify.
   b. Why is this form not used in practise? (Hint: think about the implementation.) (Tanenbaum, problem 2.13)

Extra Credit

9. (*5 points*) When a message is sent to a sleeping process in MINIX, the procedure ready is called to put that process in the proper scheduling queue. This procedure starts out by disabling interrupts. Explain. (Tanenbaum, problem 2.30)
10. (*5 points*) The MINIX procedure mini_rec contains a loop. Explain what it is for. (Tanenbaum, problem 2.31)
11. (*5 points*) Can the precedence graph below be implemented using *parbegin* and *parend*?