BACI Projects for an OS Course

1. **A's and B's:** For the following program outline in Ben-Ari Concurrent Pascal,

```
PROGRAM As_and_Bs;

VAR
    { semaphore declarations }

PROCEDURE A;
BEGIN (P's and V's only) END;

PROCEDURE B;
BEGIN (P's and V's only) END;

BEGIN
    { semaphore initializations }
    COBEGIN A;A;A;B;B; COEND;
END.
```

   complete the program using *general* semaphores so that the processes ALWAYS terminate in the order A (any copy), B (any copy), A (any copy), A, B. In addition to the program, hand in the results of FIVE executions of the program using the -apt option. Include an explanation (or proof, if you prefer) of why the program works like it should.

2. **Binary Semaphores:** Repeat problem 1 using only BINARY semaphores. In addition, you can use assignment, incrementing, and decrementing auxiliary INTEGER variables, along with IF-(testing INTEGER expressions)-THEN-ELSE. Consider why this problem can not be solved without counting variables.

3. **More A's and B's:** Repeat the A and B problem using four concurrent processes (A, A, A, and B) such that they terminate in the order A (any copy), B, A, A.

4. **Even More A's and B's:** Repeat the A and B problem using eight concurrent processes (A, A, A, A, B, B, B, B) such that they terminate in the order AABABABB. **Tough one!**

5. **Busy Waiting versus Semaphores:** Using the program developed in the general semaphore project above (ABAAB), consider the performance effect of its execution with your fair and your unfair (random) semaphore implementations. That is, calculate the total number of busy waiting loops that occur in each implementation. Compare the performance of these two semaphore implementations with a busy waiting implementation, i.e., an implementation that uses the exchange operation for synchronization. In each case, use a large number of executions (say, 1000) to obtain better statistics. Discuss your results, explaining why one implementation is preferred over another.