Problem 1 (0 Points)
Consider the following parallel program:

```c
int X; // Global (shared)
void inc() { int n; for (n=0; n<100; n++) if (X<10) ++X; }
void main() { X=0; parbegin inc(); inc() parend; print(X); }
```

The construct `parbegin S1; S2; ... parend` with statements S1, S2, ... means that the statements can execute in parallel subject to any synchronization primitives. In the above case, there is no synchronization between the two instances of the `inc` function.

Determine the smallest and largest value of the shared variable X that will be printed and explain how you arrived at this answer. Assume processes can execute at any relative speed and that a value can be incremented/decremented after it has been loaded into a register.

Problem 2 (0 Points)
This problem considers Peterson’s 2-process algorithm given in class.

a) Explain how the algorithm prevents one process from monopolizing the critical section; i.e., prevent starvation?

b) Explain how the algorithm guarantees freedom from deadlock?

Problem 3 (0 Points)

a) Give an algorithm that uses the `TestAndSet` hardware mutual exclusion instruction to update a shared variable X in a consistent manner.

b) Suppose that there are 1,000 processes that potentially can update X, but only a few (2 or 3) concurrently want to update X. How does the `TestAndSet` instruction speed-up the updating of X compared to a software-only algorithm (e.g., Peterson’s algorithm)?
Problem 4 (0 Points)

Consider the following algorithm and assume that we have created \( N \) processes.

Shared Variables:

\[
\text{Semaphore} \quad X = N, \quad Y = 1; \\
\text{Semaphore} \quad Z[N] = 0; \quad \text{// array of \( N \) semaphores initialized to 0} \\
\text{int} \quad n = 0, \quad w = 0;
\]

Process \( i \):

\[
\text{int} \quad \text{next} = (i+1) \mod N; \\
\text{do \ forever \{} \\
\quad \text{Wait}(X); \\
\quad \ldots \text{ Compute } \ldots \\
\quad \text{Wait}(Y); \\
\quad n = n + 1; \\
\quad \text{if} \ (n \geq N) \{ \\
\quad \quad n = 0; \quad \text{// Place A} \\
\quad \quad w = i; \\
\quad \quad \text{Signal}(Y); \\
\quad \quad \text{Signal}(Z[\text{next}]); \\
\quad \} \quad \text{else \{} \\
\quad \quad \text{Signal}(Y); \\
\quad \quad \text{Wait} (Z[i]); \\
\quad \quad \text{if} \ (\text{next} \neq w) \quad \text{Signal}(Z[\text{next}]); \\
\quad \} \\
\quad \text{Signal}(X); \\
\}
\]

a) What is the purpose of each of the semaphores \( X, Y, \) and \( Z[i] \)?

b) What would be the effect of deleting the statement labeled Place A from the algorithm?
Problem 5 (6 Points)
Consider the following software-only mutual exclusion algorithm:

```java
boolean blocked[2];
int who;
void P (int id) {
    while (TRUE) {
        blocked[id] = TRUE;
        while (who != id) {
            while (blocked[1-id]) who = id;
        }
        // --| critical section goes here |--
        blocked[id] = FALSE; // exit section
        // --| ... other processing ... |--
    }
}
void main () {
    blocked[0] = blocked[1] = FALSE;
    who = 0;
    parbegin ( P(0), P(1) );
}
```

a) If the processes execute at the same rate as much as possible, in what order will the processes enter the critical section.

b) Summarize how the algorithm attempts to guarantee mutual exclusion in the general case.

c) The algorithm contains some flaws. Give an example where the algorithm exhibits starvation.

d) Give an example where the algorithm exhibits livelock and explain why your example exhibits livelock.

e) In what sense is the algorithm speed-sensitive?

Problem 6 (4 Points)
Barrier synchronization between N processes works as follows:

- A counter is initialized to N, the number of processes participating in the barrier.
- The first N − 1 processes arriving to a barrier should wait.
- The Nth process arriving to a barrier should unblock the N − 1 waiting processes so that all N processes can continue with the rest of the program.

a) Give a barrier synchronization algorithm that uses the TestAndSet hardware instruction and busy waiting. Assume that there are no synchronization functions except the TestAndSet instruction. Note that the algorithm needs to perform a barrier synchronization between N processes only once. Also, assume that there are no other synchronization primitives available.

b) Explain how your algorithm works.