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() pend; print(X); }
```

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 (5 Points)
Consider the following software approach to mutual exclusion between four processes numbered 0, 1, 2, and 3. `num[4]` and `testing[4]` are shared global arrays (integer and boolean respectively).

```c
Process i:
1  testing[i] = TRUE;
2  num[i] = 1 + max(num[0], ..., num[3]);
3  testing[i] = FALSE;
4  for j=0 to 3 {
5      while (testing[j]) { ... do nothing ... }
6      while ((num[j] != 0) and ((num[j], j) < (num[i], i))) {...do nothing...}
7     } // NOTE: (a,b) < (x,y) if (a<x) OR ((a==x) AND (b<y))
8     ... critical section ...
9  num[i] = 0;
```

a) If the four processes execute at the same rate as much as possible, in what order will the four processes enter the critical section? For example, all processes will begin and finish executing Line 2 at the same time. Explain.

b) What would be the effect of omitting Line 5?

c) How does the algorithm guarantee mutual exclusion?
Problem 4 (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.

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 5 (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 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 $N$th 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.

Problem 7 (0 Points)

Consider the following algorithm and assume that we have created $N$ processes.

Shared Variables:
- Semaphore $X = N$, $Y = 1$;
- Semaphore $Z[N] = 0$; // array of $N$ semaphores initialized to 0
- int $n = 0$, $w = 0$;

Process i:

```c
int next = (i+1) mod N;
do forever {
  Wait(X);
  ... Compute ...
  Wait(Y);
  n = n + 1;
  if (n >= N) {
    n = 0;  // Place A
    w = i;
    Signal(Y);
    Signal(Z[next]);
  } else {
    Signal(Y);
    Wait (Z[i]);
    if (next != w) Signal(Z[next]);
  }
  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 8 (2 Points) EXTRA CREDIT

Consider two processes (0 and 1) that repeatedly execute Peterson’s two-process algorithm. But suppose that we run the algorithm on a single-CPU with a time-shared operating system where every process is given an equal time slice. Let \( N_k \) be the number of times that process \( k \) completes its critical section over a very long time period. And let \( T_k \) be the execution time of process \( k \).

a) Explain how it is possible that the ratio \( (T_0/T_1) \) is much larger for large time slices than for small time slices if we assume the following:

   - The two processes have identical CPU execution behavior outside of the critical section and make no system calls.
   - There are no other processes.
   - The operating system kernel does not execute except to preempt a process at the end of a time slice (because no system calls are executed and there are no administrative tasks).
   - All processes have the same priority and are given the same amount of time (the quantum or time slice) before being preempted by the other process.

   Explain.

b) There is a possibility of priority inversion when using Peterson’s algorithm and one process has a higher priority than the other. Explain what that means here.

Problem 9 (4 Points)

The last bank teller example given in class has the following features;

- There are \( M \) tellers and \( N \) customers.
- The bank lobby has a capacity of \( K \) customers.
- \( M \) tellers service customers one at a time from a common FIFO queue.
- A teller tells customers when they are ready before a customer can come to the teller window.
- Customers arriving to a full lobby return after a random delay.

Here is the algorithm given in class:

```c
Global Variables:
    Semaphore tRdy = 0, cRdy = 0, tDone = 0;
    int n;
    Semaphore nLock = 0;

Process customer (int i) {
    do forever {
        ... Random Delay ...
    } until [[ n < 20 ]]
    [[ n = n+1; ]]
    Wait(tRdy);
    Signal(cRdy);
}
Process teller (int i) {
    do forever {
        Signal(tRdy);
        Wait(cRdy);
    }
    ... Serve customer ...
    Signal(tDone);
}
```
... Get service ...
}  
Wait(tDone);
[[ n = n-1; ]]  // [[ ]] means Wait(nLock); ... Signal(nLock);
... Leave bank ...
}

The bank has decided to add the following new features:

- There will be one extra teller that will service preferred customers.
- Preferred customers will line up in their own line (separate from the other customers) in the lobby.
- The *preferred-customer* teller will service any customer to completion if there is no waiting preferred customer but will always give preference to a preferred customer when selecting the next customer.
- All other tellers will service customers from the preferred line if there are no other customers.

a) Indicate what changes need to be made to the original algorithm to capture the new features.

b) Briefly summarize and explain the changes you made.