Written Assignment #2
EECS497.32
Grace Livingston

1. What are the two main methods to protect shared data?

Answer: semaphores and mutex

2. Is this function reentrant? If not, how do you make it reentrant?

```
int Total_Errors;
void CountErrors ( int New_Error)
{
Total_Errors = New_Error + Total_Errors;
}
```

Answer:

This function is not reentrant because the addition operation is not atomic, and the function performs this on a module-scoped variable. Each call to the function would perform multiple operations depending on its value or updating its value without coordination with other calls to this function, so if the variable is updated by one call while another call is operating on its previous value, the ongoing call will overwrite the new value with one that's based on the old value.

To make this function reentrant, adding a semaphore that the function can take before operating on it. The semaphore will block a Task while another Task has acquired it, and will prevent other Tasks from running while the current Task has it held.

3. For each of the following situations, which one of the two shared data protection methods discussed in class works best?
   a) Task A and Task B share an int array, and each of these tasks update many elements inside the array in an RTOS environment.
   b) The Task code shares a single char variable with one of the interrupt service routines(ISR).

Answer:

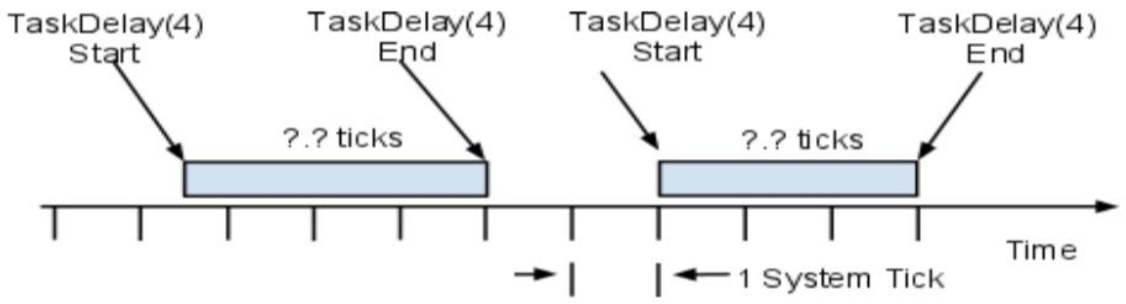
   a) Since the array requires multiple elements to be updated at once, locking each element would create several additional lock objects that would have a higher chance of contention, would consume more memory, and require additional atomic operations. Having a single semaphore for the array, though, would allow each task to acquire the single semaphore, perform all read or update operations at once, and then release the semaphore. This would be a single atomic operation, and would ensure the consistency of the entire array without a correspondingly large footprint of per-element locks. Per-element locks would still require a full-array lock if the array itself must be modified.
   b) For this situation, masking the interrupt while operating on the shared variable in the Task would be the most effective solution. This would prevent that interrupt from reading or updating the character while the Task is accessing it, and while the ISR is running, the Task wouldn't be anyway, which effectively prevents the Task from reading or updating the variable concurrently.

4. What are the two rules an interrupt service routine(ISR) in a RTOS environment must follow that does not apply to the Task code.

Answer:

   a. The ISR must save the Status Register on entry
   b. The ISR must restore the Status Register on exit

5. For how long (in ticks) will the TaskDelay function be blocked in the following two cases?



Answer:

1. In the first case, since TaskDelay is called between ticks, it will remain blocked for the minimum of 4 ticks it's required to be blocked for, plus the fraction of the tick it's called during (*f*), so 4 + *f* ticks.
2. For the second case, since TaskDelay is called right at the beginning of a tick, the function will remain blocked only for that tick plus the 3 others of the required 4 ticks, so the total time it's blocked will be 4 ticks.