Task A

1. While they seemingly seem to run fine at the same time. Whenever the timers finish at the same time and both try to run and print their statements at the same time printing out garbage.
2. While it now longer produces garbage, if the primary task is running and the secondary task triggers it will stop task 1, meaning the output will no longer have garbage when timers intersect, but they will be a mix of the two messages. Additionally, the timer in task 1 is no longer accurate if it is interrupted by task 2 while running. This can occur because the time is using a software solution
3. It only prints out the message in secondary task, this is because without the delay secondary task is always running, and since it has a higher priority it means the first task will never get a free chance to run.

Task B

1. The binary semaphore prevents the task from running continuously by having the thread wait for the semaphore to give the signal, this way the program waits until the signal is given. Otherwise, it would run an if statement for ever so see what happens. This is considered good practice because it minimizes overhead, is more efficient and allows other threads to use the CPU while this thread waits for a signal.
2. The design choice I made for this design is to have the system ok task have the lowest priority, because otherwise there is the possibility for the ‘system ok’ task to pre-empt the button/led task. To solve this, both the buttons have a higher priority than the System ok task