**Communication Challenge 2: FreeRTOS**



A screenshot of a computer

Description automatically generated

*Circuit used: red wires are to simulate power to the component that is connected to its designated pin. The black wires are to simulate being connected to the GND pin. Each LED represents a sprinkler and they are connected to my ESP32 as show in the above circuit.*

Student: Johnson Domacasse

Date: 11 Okt 2023

Student#: 4471709

Teacher: Hans van Heumen

## Introduction:

## Procedure:

I begin by setting up what I believe to be my circuit for this entire challenge. See above circuit diagram. I have set up the pins in the

### Setup:

|  |  |
| --- | --- |
| **COMPONENT** | **PIN** |
| LED\_1\_PIN (Sprinkler 1) | D14 |
| LED\_2\_PIN (Sprinkler 2) | D26 |
| LED\_3\_PIN (Sprinkler 3) | D21 |
| LED\_4\_PIN (Sprinkler 4) | D22 |

*Table 1: Hardware Pins*

### Design choices thread:

Based off of the requirements that were given to me, I came up with the following key conclusions:

* Each sprinkler own designated LED and a function to control turning the LEDs on and off.
* Store the operations of scenes. They are hardcoded and stored in my application
* Make a function to execute a scene, by reading these operations of the hardcoded stored scenes and perform said operations.
* I need to test this function. First without threads.
* Make on thread, that does nothing but read the UART repeatedly. If it receives a number (in this case representing a scene), then end echo this number back to my laptop. This thread should never end.
* In that same thread when a valid number of a Scene is received from the user, start another thread that performs the operations of that Scene by calling the function executeScene that was defined before.
* Make sure that that the thread ends itself in a controlled way when the Scene is done.
* Test my program with multiple Scenes running at the same time. For example: first type ‘1’ and a while later type ‘2’; Scene 1 and 2 should then run together, which you can check by looking at the LED’s.

### Thread implementation:

Now that I knew my requirements I began to work. I first defined my LED pins to simulate the sprinklers. I then made the predefined scenes. These are hardcoded arrays with messages to simulate its behaviour. I chose to do it like this because this is one of the methods I am most familiar with. I have made a simple function to control the state of the LED by providing it with the LED pin and the state I want the LED to be in. I defined another function to execute the scenes by providing it with the array and the size of the array. It would then perform the operations of the given array. Finally in my loop I call that function 3 times, to test all 3 scenes. The only issue I see with this implementation is that I need to implement delays in the main function for it to have my desired behaviour.

Now I will work on the thread part of this implementation.

I tested it in a way so that a user can input values. So I tested a case, where an invalid scene is provided, an invalid character is provided and the correct one. The special problems I had was that the LEDs do turn on per scene but they do not continue with the array. For example, if 1 was provided, the execute scene task would call the execute scene function and the execute scene function would receive array 1 and its size. The special problem that I encountered was that when I call the same scene twice, it would not execute the scene. I am assuming this is due to the lack of mutex usage. More will be researched in the mutex assignment.

### Design choices mutex:

Based off of the previous thread requirements and the ones that were now given to me, I have come with the following conclusions:

* Make the program in a way so that the scenes do not interfere with eachother if both scenes are using the same sprinkler. I need to implement this by adding one mutex per sprinkler. (acquire the mutex before start and release mutex at the end of operation.)
* Make sure a scene doesn’t execute twice. If I use 2 scenes then they will run in parallel because, each scene has it’s own thread. But if I use a singular scene then the second operation of the scene must wait for the first to finish before running a second time.

Each scene must have 1 mutex. Before a thread executes the first step of a scene, it must acquire that scenes mutex and after the last step it will release it.

### Mutex implementation:

I took the previous thread assignment and from there it was rather simple what needed to be done. based on the design choices above, I have made one mutex for each LED. This will ensure that if 2 scenes use the same LED there will be no issues for the scene being called twice. It will need to take the mutex first, which it can’t since the first one has the mutex of that LED. I have done this for every LED. Now there is no more unpredictable behaviour with the LEDs if I have two scenes running with the same LED.

Now I needed to make sure that when I call a scene twice, that it will run twice. Initially, it wouldn’t run twice. The second one would be ignored. So I thought but firther investigation made me realize it is still being called, but they run simultaneously. This was a problem that I solved with the mutex for the scenes themselves. So in my scene execution thread, I included two lines to take and give the mutex of that specific scene. I take the mutex of the scene, then call the function to execute the scene, then I give the mutex back before deleting the thread.

In the end, if a scene is being called twice, it will now run the first time, then when it finishes, it will run the second time.

### Design choices queue:

Based off of the previous thread and mutex requirements and the ones that were now given to me, I have come up with the following conclusions:

* When a new thread is created to execute a Scene, the steps of that Scene should be sent to that thread through a Queue (so the thread will not access the Scene-array directly anymore, but get the steps from the Queue).
* Test if my program is now completely thread safe? In other words, works correct all the time.

### Queue implementation:

Simply put I would need to repeat the same logic I did to fix the issues in the thread section of this assignment with a queue instead of a mutex. So I took the same code I had in the thread section and started implementing based on the choices above.

(EXPLAIN HOW I IMPLEMENTED QUEUE IN STEP 1)

(ANSWER QUESTION 2)

(ANY SPECIAL PROBLEMS ENCOUNTERED)

## Results:

(talk about not iterating through the array correctly)

(talk about testing cases)

And how I fixed em.

## Conclusions: