Assignment 3 Report

Namn: Kenan Maslan (km222ug)
Kurs: **1DT302**
Datum: 2023-06-13

Description of the project:

In this project, I am using a variety of hardware to test pyRTOS. All of the “tasks” will run at the same time, and *need* to work all the time. It needs to be predictable and reliable.

The first component I have is a DHT11 Temp/Humidity sensor. This sensor measures the temperature digitally and sends the readout to the pico.
The 2nd component I am using is the built in WiFi chip. This chip is used to upload the temperature in this project using MQTT.
The 3rd component is a button, and its purpose is to trigger a printout of all the current processing times for the different tasks.
The 4th is not a component, but it's still a task. It's the function that prints out the processing times.

Real time requirements:

The real time requirements I have for this project is that the button should work at any moment, and the button should not interrupt, it should just print out the processing times and alongside we want the temperature to also work even if we are constantly pressing the button without any delays. (That is measuring and uploading). The reason RTOS is necessary for this is because we want to make sure everything works together with reliability and good timing. Also, when we see the results, we will easily understand the main reason that RTOS is superior for these types of systems.

Implementation process:

I implemented this project by starting in the RTOS end actually, and the reason was so that we could easily just remove the RTOS library and see how it works without it. Both the RTOS and the non-RTOS versions are written in CircuitPython, but the RTOS version is using the pyRTOS library, which essentially replicates an RTOS but is still not a “true” RTOS since its still running through CircuitPython. For the implementation in RTOS i started going through each task at a time until i had all four ready, then i scheduled them and started the program.

Test protocol:

I will test and check all of the processing times for each task. This is how much time it takes for the task to run and finish.

I will also check the timing in adafruit. Adafruit logs the current time down to the second, and using this we can check what intervals the temperature is being uploaded in.

Also, I will document other differences I observe that are not measured, like responsiveness and stability..

Result:

When comparing the RTOS and non-RTOS implementation, I observed four main differences.

The first difference we see is the processing time for the temperature sensor is much lower for the RTOS version, and the reason is because RTOS runs more efficiently than the non-RTOS code.

non-RTOS processing times:

```
Process time for print: 0.0 seconds
Process time for temp: 0.271973 seconds
Process time for button: 0.0 seconds
Process time for uplink: 0.00390625 seconds
```

RTOS processing times:

```
Process time for print: 0.0
Process time for temp: 0.000976563
Process time for button: 0.0
Process time for uplink: 0.00390625

25
```

The 2nd difference we see is that the button responds much quicker in RTOS. The reason for that is because we are scheduling the tasks and running them side by side, whilst in the non-RTOS code we are just running everything in one big loop. I tried solving this for the regular code, which I did, but that brings us to the third problem.

The 3rd difference is that the reliability/stability of the RTOS implementation is much better. When we constantly check for button presses in the non-RTOS code, we run into pystack errors. This can be avoided by not running the button check constantly, but in that case the button needs to be held in to trigger it when the code checks for it. Another solution is to not check as often, but in this case we do not meet the real time requirements.

The last difference is the timing of the temperature readout and upload to adafruit. As we can see in the result below, the RTOS has much better continuity compared to the regular code that is out of time (sometimes 1 second, sometimes 2, sometimes 3). The temperature should come every 2 seconds, not in unpredictable time intervals.

non-RTOS time intervals between temperature logs (in Adafruit):

2023/06/14 10:48:28AM	25
2023/06/14 10:48:26AM	25
2023/06/14 10:48:23AM	25
2023/06/14 10:48:22AM	25
2023/06/14 10:48:19AM	25
2023/06/14 10:48:16AM	25

RTOS time intervals between temperature logs (in Adafruit):

2023/06/14 10:46:21AM	25
2023/06/14 10:46:19AM	25
2023/06/14 10:46:17AM	25
2023/06/14 10:46:15AM	25
2023/06/14 10:46:13AM	25
2023/06/14 10:46:11AM	25
2023/06/14 10:46:09AM	25
2023/06/14 10:46:07AM	25
2023/06/14 10:46:05AM	25
2023/06/14 10:46:03AM	25

Benefits and limitations:

The benefits for using an RTOS is explained in the result itself. We got better timing, lower delays, better stability/reliability and overall code that was easier to read and also to program. This last argument may not be applicable to a “real” rtos, since I used a library for python, and C code may be harder to code in.

The drawbacks of this particular RTOS is that it's not really a real RTOS, it really depends what your interpretation of a RTOS is, but for me the operating system part is not there. While we got good results from the library, we are still running CircuitPython instead of a RTOS that is closer to the hardware. A real RTOS would probably yield better results, but i could not get a RTOS to install or work in my working environment.

The drawbacks of RTOS generally is that its more complex to work with since we are introducing an additional “layer” to the project. It also requires real time debugging techniques which are specific ways of debugging just because its RTOS we are working with. Also, when running RTOS we may run into compatibility issues if a microcontroller or component is not compatible with RTOS. Finally, switching between different RTOS often requires porting the code which also creates a challenge when using RTOS for embedded systems.