Trabalho M3

Gustavo da Silva Mafra
Escola do Mar, Ciência e Tecnologia
Universidade do Vale do Itajaí
Itajaí, Brasil
gustavo_mafra@edu.univali.br

Nathália Adriana de Oliveira
Escola do Mar, Ciência e Tecnologia
Universidade do Vale do Itajaí
Itajaí, Brasil
oliveiranathalia@univali.br

Abstract—This document is a technical report about an academic work related to the subject real time systems, including exercises and complementation of a previous work

Keywords—real time, esp32, style, threads

I. Introdução

Este trabalho é referente à disciplina Sistemas de Tempo Real, onde é dividido em duas partes, a primeira parte é a resolução de duas listas relacionadas a RTOS e a segunda é referente a uma adaptação do trabalho da primeira media da disciplina, onde é aplicado o uso de threads.

II. Metodologia

A implementação deste projeto foi desenvolvida utilizando a linguagem C, e o dispositivo ESP32 apresentado na Figura 1.

(Figura 1 – ESP32)

III. Problemática

O Trabalho da M3 consistirá em 2 partes que irão um único relatório, logo a primeira parte é a execução dos código das listas de exercícios de FreeRTOS e ESP-IDF, no caso a Lista RTOS (Exercícios 1, 2, 3, 4, 5 e 7) e Lista 2 RTOS. Deverão ser executados os experimentos com os códigos e feito uma análise de cada funcionalidade expressada em cada código - descrever a funcionalidade e análise de execução dentro do contexto de Sistema Operacional e Sistema Operacional de Tempo Real.

A segunda parte diz respeito ao desenvolvimento do código relativo ao **Exercício - Threads** (adaptação do código do primeiro trabalho para um contexto de código que seja executável no FreeRTOS). Reaproveite o enunciando do trabalho feito na M1.

- Os códigos com Pthreads está disponível na pasta Exercícios (**Exercício - Threads)**. Você deve emular a existência do problema (interrupção causado sobre identificação de pressão elevada ou problema) usando o touch sensor com interrupção.
- Poderá ser feito para um sensor em um duto gás, um sensor em um duto de petróleo e um sensor em um poço de petróleo. No caso, há uma sensor que identifica ambas as pressões no duto de gás e petróleo. Há também uma tarefa supervisora que é responsável por fazer o log de eventos (printar) na tela periodicamente. Os tempos de período de execução e apresentação pode ser os mesmos do trabalho da M1.
- A captura do tempo de execução pode ser feita usando as bibliotecas da ESP32.
- Você poderá simular a comunicação de dados (via delay) ou usar algum sistema de comunicação de dados via Wi-Fi/Bluetooth. No caso da última opção, você terá a bonificação de até 2,0 pontos em uma das avaliações da M1 ou M2 (atribuição total ou não fica a critério do professor) ou 1 ponta na Média com a menor nota.
- Deverá ser feita uma apresentação de no máximo 5 minutos demonstrando essa solução.

Identify applicable funding agency here. If none, delete this text box.

IV. Desenvolvimento

A. Lista 1

1) Execute o código hello_world_main.c na placa ESP32 e descreva as funções usadas. Informe também se há alguma função do FreeRTOS? Há diferença na criação do código para ESP32 quando comparado a outros ports?

```
Hello world!
This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision 1, 4MB external flash
Minimum free heap size: 291376 bytes
```

(Figura 2 – Execução do código hello_world_main.c)

Foi utilizado funções para exibir as informações da placa ESP, e o uso do delay. A vTaskDelay é uma função do FreeRTOS para especificar a hora em que a tarefa irá ser desbloqueada. E sim tem diferença na criação do código da ESP para os outros, pois possui suporte para Bluetooth e Wi-fi, como também a possibilidade de multiprocessamento.

2) Execute o código hello_world_task.c na placa ESP32 e responda:

```
I (294) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
Hello world!
```

(Figura 3 – Execução do código hello_world_task.c)

a) Quais são os parâmetros usados para a criação da tarefa?

- Função hello_task para mostrar o *hello world;*
- vTaskDelay é uma função do FreeRTOS para especificar a hora em que a tarefa irá ser desbloqueada;
- xTaskCreate é uma função para criar uma tarefa e adicioná-la a lista de tarefas que vão ser executadas.

b) Há formas de definir a prioridade?
Sim, pode-se definir a prioridade por parâmetro.

c) Como o port do FreeRTOS para ESP32 faz o escalonamento?
É realizado pela tarefa, que chama a função com maior prioridade, e que estão prontas para serem executadas, além de que, cada núcleo chama uma tarefa de forma independente.

3) Execute o código hello_world_and_blink_task.c na placa ESP32 e responda:

```
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
```

(Figura 4 – Execução do código hello_world_and_blink_task.c)

a) Há alguma diferença entre as duas tasks criadas?
Sim, as duas tarefas possuem o número de palavras diferentes a serem alocadas para utilizar na pilha da tarefa.

b) A função vTaskDelay() serve para qual motivo? Há diferença para as duas tasks?
A função vTaskDelay serve para especificar a hora em que a tarefa irá ser desbloqueada.

4) Execute o código task_creation_and_priority.c e responda:

```
Hello world from Task 2!
Hello world from Task 1!
Hello world from Task 2!
Hello world from Task 1!
Hello world from Task 2!
Hello world from Task 1!
Hello world from Task 2!
Hello world from Task 1!
Hello world from Task 1!
Hello world from Task 2!
Hello world from Task 1!
Hello world from Task 2!
```

(Figura 5 – Execução do código task_creation_and_priority.c)

*a)*Qual a diferença entre as funções xTaskCreatePinnedToCore() e xTaskCreate()?

- xTaskCreatePinnedToCore: é uma função para fixar uma tarefa específica em um núcleo específico, logo com isso pode-se executar duas tarefas diferentes de forma independente e de forma simultaneal usando os dois núcleos
- xTaskCreat: é uma função para criar uma tarefa e adicioná-la a lista de tarefas que vão ser executadas

b) Como é o esquema de prioridade no FreeRTOS?
Cada tarefa tem a sua determinada prioridade, logo a ordem é definida pelo escalonador, visto que, é o administrador das tarefas que irão ser executadas pelo CPU. Logo, tem-se diversos algoritmos para definir as tarefas, como por exemplo o Round Robin.

c) Quantas tarefas estão sendo executadas ao mesmo tempo? Todas têm a mesma prioridade?
Estão sendo executadas duas tarefas ao mesmo tempo, e elas tem a prioridade diferente, sendo a primeira tarefa a de maior prioridade.

5) Execute o código task_mutual_exclusion.c:

```
Mutex was created

I am Task 10

I am IDLE function0

I am Task 20

I am Task 10

I am IDLE function0

I am Task 20

I am Task 10

I am IDLE function0
```

(Figura 6 – Execução do código task_mutual_exclusion.c)

a) Como pode ser oferecido a exclusão mútua no RTOS?
A partir da operação *xSemaphoneCreateMutex*, onde é criado um mutex e é retornado um identificador para que o mutex possa ser referenciado.

*b)*Qual a diferença entre as operações taskENTER_CRITICAL(), vTaskSuspendAll() e xSemaphoreCreateMutex()?

- taskENTER_CRITICAL: é uma operação que tem ínicio e fim, sendo respectivamente *taskENTER_CRITICAL()* e *taskEXIT_CRITICAL()* que funciona para desabilitar interrupções, sendo de forma global ou com nível de prioridade de interrupção específico.
- vTaskSuspendAll: é uma operação que suspense o escalonador
- xSemaphoreCreateMutex: é uma operação que cria um mutex e faz o retorno de um identificador pelo qual o mutex criado poderá ser referenciado

7) Usando as funções vTaskSuspend() e vTaskResume() é possível implementar um monitor?
Sim, pois a função vTaskSuspend é utilizada para suspender uma tarefa e a vTaskResume serve para o retorno da tarefa. Logo tem-se a possibilidade de criar um monitor envolvendo as prioridades de tarefas e o bloqueio e seguimento das mesmas.

B. Lista 2

1) Execute os códigos touch_pad_example.c e touch_pad_int.c na ESP32 para exemplificar o uso de periféricos e responda a pergunta:

```
Touch Sensor normal mode read, the output format is:
Touchpad num:[raw data]

T0:[1505]
T0:[1505]
T0:[1505]
T0:[1505]
T0:[1505]
T0:[1506]
T0:[1505]
T0:[1505]
T0:[1504]
T0:[1503]
T0:[ 200]
T0:[ 107]
T0:[  88]
Sistema de Emergência Acinado
T0:[ 319]
T0:[ 213]
T0:[1502]
```

(Figura 7 – Execução do código touch_pad_example.c)

```
Usuario pediu para abrir a porta: 1

Usuario pediu para abrir a porta: 4

Usuario pediu para abrir a porta: 6

Usuario pediu para abrir a porta: 8

Hello World

Usuario pediu para abrir a porta: 1

Usuario pediu para abrir a porta: 4

Usuario pediu para abrir a porta: 6
```

(Figura 8 – Execução do código touch_pad_int.c)

*a)*Há diferença nos dois códigos quanto ao monitoramento do periférico touch sensor?
Sim, o código *touch_pad_example.c* monitora somente um sensor, e o código *touch_pad_int.c* monitora todos os sensores.

2) Execute o código gpio_intr_example.c na placa ESP32 e responda:

```
entry 0x40080680
I (27) boot: ESP-IDF GIT-NOTFOUND 2nd stage bootloader
I (27) boot: compile time 10:27:44
I (27) boot: chip revision: 1
I (30) boot_comm: chip revision: 1, min. bootloader chip revision: 0
I (37) boot.esp32: SPI Speed      : 40MHz
I (42) boot.esp32: SPI Mode       : DIO
I (46) boot.esp32: SPI Flash Size : 4MB
I (51) boot: Enabling RNG early entropy source...
I (56) boot: Partition Table:
I (60) boot: ## Label            Usage          Type ST Offset   Length
I (67) boot:  0 nvs              WiFi data        01 02 00009000 00006000
I (75) boot:  1 phy_init         RF data          01 01 0000f000 00001000
I (82) boot:  2 factory          factory app      00 00 00010000 00100000
I (90) boot: End of partition table
I (94) boot_comm: chip revision: 1, min. application chip revision: 0
I (101) esp_image: segment 0: paddr=00010020 vaddr=3f400020 size=0755ch ( 30044) map
I (120) esp_image: segment 1: paddr=00017584 vaddr=3ffb0000 size=028e4h ( 10468) load
I (125) esp_image: segment 2: paddr=00019e70 vaddr=40080000 size=061a8h ( 25000) load
I (137) esp_image: segment 3: paddr=00020020 vaddr=400d0020 size=14aech ( 84716) map
I (168) esp_image: segment 4: paddr=00034b14 vaddr=400861a8 size=0537ch ( 21372) load
I (177) esp_image: segment 5: paddr=00039e98 vaddr=50000000 size=00010h (    16) load
I (183) boot: Loaded app from partition at offset 0x10000
I (183) boot: Disabling RNG early entropy source...
I (198) cpu_start: Pro cpu up.
I (198) cpu_start: Starting app cpu, entry point is 0x40081094
0x40081094: call_start_cpu1 at C:/Users/6845207/esp/esp-idf/components/esp_system/port/cpu_start.c:141

I (0) cpu_start: App cpu up.
I (213) cpu_start: Pro cpu start user code
I (213) cpu_start: cpu freq: 160000000
I (213) cpu_start: Application information:
I (217) cpu_start: Project name:     hello-world
I (223) cpu_start: App version:      1
I (227) cpu_start: Compile time:     Nov 25 2021 10:27:34
I (233) cpu_start: ELF file SHA256:  8a749a6db290e8d6...
I (239) cpu_start: ESP-IDF:          GIT-NOTFOUND
I (245) heap_init: Initializing. RAM available for dynamic allocation:
I (252) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (258) heap_init: At 3FFB3318 len 0002CCE8 (179 KiB): DRAM
I (264) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (270) heap_init: At 3FFE4350 len 00001CB0 (111 KiB): D/IRAM
I (277) heap_init: At 40088524 len 00014ADC (82 KiB): IRAM
I (284) spi_flash: detected chip: generic
I (288) spi_flash: flash io: dio
I (293) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
I (302) Touch pad: Initializing touch pad
I (2512) Touch pad: test init: touch pad [0] val is 1513
I (2512) Touch pad: test init: touch pad [1] val is 0
I (2512) Touch pad: test init: touch pad [2] val is 0
I (2512) Touch pad: test init: touch pad [3] val is 1384
I (2522) Touch pad: test init: touch pad [4] val is 1533
I (2522) Touch pad: test init: touch pad [5] val is 1680
I (2532) Touch pad: test init: touch pad [6] val is 1655
```

(Figura 9 – Execução do código touch_pad_int.c)

a) Há diferença no periférico quanto ao uso do periférico para o exemplo anterior? O uso do FreeRTOS é necessário?
Sim, pois no exemplo anterior era usado o sensor, e nesse é utilizado o botão. O uso do FreeRTOS é necessário pois tem as operações de *xTaskResumeFromISR()* e *vTaskSuspend()*, onde é possível obter controlar sobre a tarefa.

3) Execute o código library_and_timers.c na placa ESP32 e responda:

```
Usuario pediu para abrir a porta: 8

Usuario pediu para abrir a porta: 10

Hello World

Usuario pediu para abrir a porta: 1

Usuario pediu para abrir a porta: 4

Usuario pediu para abrir a porta: 6

Usuario pediu para abrir a porta: 8
```

(Figura 10 – Execução do código library_and_timers.c)

a) O temporizador usado é de sistema ou de hardware?
O temporizador é de hardware, visto que, tem-se a função *esp_timer_get_time()*, onde é possível obter o tempo em milisegundos.

b) Há necessidade de mexer em algum arquivo para modularizar o código?
Sim, houve a necessidade de implementar uma biblioteca com as funções do exercício anterior para realizar a execução corretamente.

c) Os temporizadores em software são melhores do que usado no exemplo?
Não, pois os temporizadores em hardware possuem mais precisão.

4) Execute o código queue_and_events_group.c e responda:

```
Item recebido: 0
Item nao recebido, timeout expirou!
Item nao recebido, timeout expirou!
Item recebido: 1
Item nao recebido, timeout expirou!
Item nao recebido, timeout expirou!
Item recebido: 2
Item nao recebido, timeout expirou!
Item nao recebido, timeout expirou!
Item recebido: 3
Item nao recebido, timeout expirou!
Item nao recebido, timeout expirou!
Item recebido: 4
Item nao recebido, timeout expirou!
Item nao recebido, timeout expirou!
Item recebido: 5
Item nao recebido, timeout expirou!
Item nao recebido, timeout expirou!
Item recebido: 6
```

(Figura 11 – Execução do código queue_and_events_group.c)__

a) Qual a diferença entre usar Mutex e usar Queus e Events Group?

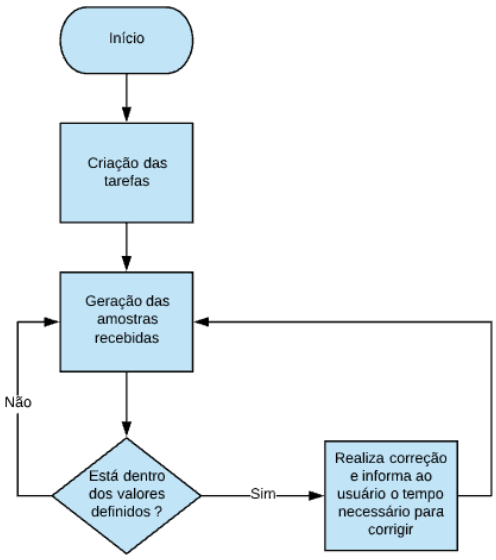
- Mutex: é um bloqueio utilizado para permitir que apenas um segmento acesse uma parte do código por vez;
- Queus: é uma fila, e quando é criada retorna um identificador para poder enncontra-lá posteriormente;
- Events Group: é um grupo de eventos que qando é criado retorna um identificador para referenciar posteriormente.

b) Posso implementar uma abordagem parecida com monitor com Events Group?
Sim, pois como o monitor bloqueia e libera atividades. Logo o Events Group, indica qual atividade foi e será realizada.

C. *EXERCÍCIO – THREADS*

Para a realização desta parte do trabalho, foi implementado a biblioteca *sys/time.h* para captar o tempo de execução de cada sensor. Para gerar interrupções, foi necessário o uso de números randômicos, e para simular os tempos de reparo tem-se tempos randômicos dentro da premissa do trabalho da primeira média. Sendo 300 ms para problema de mal funcionamento no poço e uma contramedida em até 200 ms para pressão instável na tubulação.

Dessa forma, o fluxograma da Figura 12 demonstra o funcionamento do código, sendo que, é aplicável para os sensores de gás e petróleo e também para o sensor do poço.

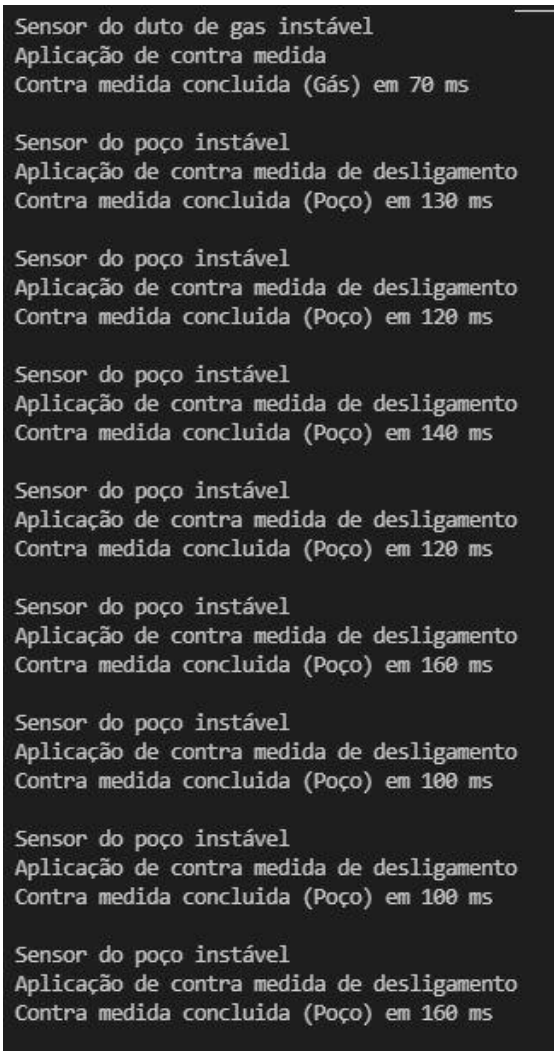


(Figura 12 – Fluxograma desenvolvido)

Na Figura 13 e Figura 14, tem-se os resultados obtidos após a simulação, onde é demonstrado os momentos de instabilidade.

```
Start ->

Sensor do poço instável
Aplicação de contra medida de desligamento
Contra medida concluida (Poço) em 110 ms

Sensor do duto de oleo instável
Aplicação de contra medida
Contra medida concluida (Óleo) em 59 ms

Sensor do duto de gas instável
Aplicação de contra medida
Contra medida concluida (Gás) em 60 ms

Sensor do poço instável
Aplicação de contra medida de desligamento
Contra medida concluida (Poço) em 190 ms

Sensor do poço instável
Aplicação de contra medida de desligamento
Contra medida concluida (Poço) em 110 ms

Sensor do duto de oleo instável
Aplicação de contra medida
Contra medida concluida (Óleo) em 140 ms

Sensor do duto de gas instável
Aplicação de contra medida
Contra medida concluida (Gás) em 140 ms

Sensor do duto de oleo instável
Aplicação de contra medida
Contra medida concluida (Óleo) em 60 ms

Sensor do duto de gas instável
Aplicação de contra medida
Contra medida concluida (Gás) em 60 ms
```

(Figura 13 – Início da execução do trabalho)



(Figura 14 – Execução em outro momento)

V. Conclusão

Em relação as listas de exercícios, foi possível executar todos as atividades, assim como responder todas as perguntas, a partir da execução e pesquisas no site do FreeRTOS [1].

E a segunda parte do trabalho, teve-se a adaptação do código do trabalho da primeira media, utilizando as funções da ESP32. Com isso, tem-se a demonstração da execução do código, que indica que os tempos definidos no trabalho foram respeitados.

VI. Referências

[1] FreeRTOS – Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensios. https://www.freertos.org/.

[2] FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions. https://www.freertos.org/.