

Lista de Exercício

1st Suzana Almeida Coelho
Universidade do Vale do Itajaí
Itajaí, Brasil
suzana.coelho@edu.univali.br

2nd Paola Oliveira Ribeiro
Universidade do Vale do Itajaí
Itajaí, Brasil
paolaribeiro@edu.univali.br

I. INTRODUÇÃO

Exercício realizado como atividade complementar referente a M3, da disciplina de Sistemas de Tempo Real, com o professor Felipe Viel.

Link de acesso a apresentação: <https://youtu.be/rBlkUxntQcg>

Link do repositório no Git: https://github.com/paolaribeiro/Trabalho_M3_RTS.git

II. LISTA RTOS 1

1) Execute o código `hello_word_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?

A maior parte das informações que foram utilizadas não pertenciam ao FreeRTOS, faz uso de funções mais dedicadas do ESP sendo a função `vTaskDelay` uma função do FreeRTOS que é exclusiva da ESP32. A função `print` é uma função de porte para qualquer aplicação, ou seja, pode ser tanto da ESP quanto do FreeRTOS. A função `app_main` não é uma task mas sim uma função chamada por uma task que é chamada no boot do sistema. Sim há diferença na criação dos código para ESP32, as funções `esp_chip_info_t` e `esp_chip_info` são características da ESP.

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

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

Para a criação da tarefa usa-se o parâmetro `xTaskCreate` onde é enviado o nome da função que representa o trecho de código da task, o nome dela, a prioridade e não passa nenhum parâmetro.

b) Há formas de definir a prioridade?

Sim, é uma necessidade no FreeRTOS pois é um sistema em tempo real.

c) Como o port do FreeRTOS para ESP32 faz o escalonamento?

O escalonamento é feito como no próprio Vanilla (versão do FreeRTOS) só que é simplificado pelo SMP.

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

a) Há alguma diferença entre as duas tasks criadas?

Sim, pois uma task tem como objetivo printar valores na tela e a outra é acender o LED, também existem diferenças de execução onde uma recebe mais memória do que a outra.

b) A função `vTaskDelay()` serve para qual motivo? Há diferença para as duas tasks?

A função `vTaskDelay` serve para determinar o tempo de uma atividade e outra. A diferença é que a task que printa tem um delay menor do que a task que acende o LED.

4) Execute o código `task_creation_and_priority.c` e responda:

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

A função `xTaskCreatePinnedToCore()` é própria da espressif é similar a função `xTaskCreate()` porém ela permite que tenha afinidade com o SMP, que possibilita pinagem do core.

b) Como é o esquema de prioridade no FreeRTOS?

Na documentação do FreeRTOS encontra-se que o esquema de prioridade é feito com os valores maiores para maior prioridade.

c) Quantas tarefas estão sendo executadas ao mesmo tempo? Todas têm a mesma prioridade?

Apesar de possuir duas tasks bem definidas, a função `app_main` faz parte da task criada pelo boot do sistema. Assim possui 3 Tasks e elas não possuem a mesma prioridade.

5) Execute o código `task_mutual_exclusion.c`:

a) Como pode ser oferecido a exclusão mútua no RTOS?

A exclusão mútua é oferecida pela operação `xSemaphoreCreateMutex`, nela é criado um mutex e é retornado um identificador para que o mutex possa ser referenciado e excluído.

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

`taskEnter_CRITICAL` - é uma operação que possui início e fim, sendo que para o início `taskEnter_CRITICAL()` e `taskEXIT_CRITICAL()` para o fim e elas funcionam para desabilitar interrupções.

`vTaskSuspendAll` - essa operação vai suspender o escalonador.

`xSemaphoreCreateMutex` - operação que vai criar o mutex e fazer o retorno de um identificador pelo qual o mutex criado vai ser referenciado.

7) Usando as funções `vTaskSuspend()` e `vTaskResume()` é possível implementar um monitor?

Sim, pois utilizando a função `vTaskSuspend` que suspende uma task e espera até que outra função chame a `vTaskResume` que retorna a executar, assim é possível criar um monitor envolvendo as prioridades de tarefas e o bloqueio e seguimentos das mesmas.

III. LISTA RTOS 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:

a) Há diferença nos dois códigos quanto ao monitoramento do periférico touch sensor?

Existe, pois um dos códigos funciona com interrupção e o outro não. Enquanto o `touch_pad_int` fica esperando a interrupção, o `touch_pad_exemple.c` fica sempre tendo que olhar se ocorreu ou não uma interrupção.

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

a) Há diferença no periférico quanto ao uso do periférico para o exemplo anterior? O uso do FreeRTOS é necessário?

Sim, pois em cima usamos uma característica de sensor, e aqui usamos para gpio no sentido geral. Na questão um utilizamos touch sensor com um tipo de interrupção associada a ele, de forma mais específica e na questão atual utilizamos uma interrupção mais genérica da ESP32.

Para uso das interrupções o FreeRTOS não é muito necessário, porém para questão de dinamicidade ele precisa ser utilizado, pois é quem dá o suporte.

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

a) O temporizador usado é de sistema ou de hardware?

É um temporizador em hardware pois pertence a ESP32.

b) Há necessidade de mexer em algum arquivo para modularizar o código?

Sim, precisa ser mexido nos arquivos de lista, como o `SRCS` e o `INCLUDE_DIRS`.

c) Os temporizadores em software são melhores do que usado no exemplo?

Não. Eles são do porte oficial do FreeRTOS, sua vantagem é que o código fica mais fácil de ser portado de um dispositivo para outro, mas a precisão fica comprometida no código que está sendo executado na ESP32, pois o temporizador de hardware é sempre mais preciso que o de software.

4) Execute o código `queue_and_events_group.c` e responda:

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

Mutex funciona como um bloqueio, permitindo que somente um produtor ou consumidor acesse uma determinada parte do código de cada vez.

Queues e o Events Group trabalham com troca de mensagens, podendo parar de utilizar várias globais e adicionando dentro das tasks códigos que permitam a troca de mensagem.

No Queues possui uma operação de sends e receive basicamente, onde é preenchido as filas e permitido que elas sejam lidas no outro lado onde vai ser efetuada a execução das tarefas. Existe prioridade de execução, tarefas de alta prioridade podem ler a fila de troca de mensagens antes das de baixa prioridade

Event Group você vai trabalhar por bit, podendo criar uma lista de eventos existentes, dentro dos bits da variável, cada bit do valor pode ser associado a um evento, podendo ter vários eventos, porém ele tem a necessidade de se trabalhar com máscaras de códigos, movimentando os bits para a esquerda e para direita e fazer a utilização do AND e/ou OR para fazer as operações lógicas entre um número e outro, caso o valor bater, você tem um evento acontecendo que é do seu interesse. Toda tarefa que faz parte do grupo de evento pode acessar a lista de execução e quem chegar primeiro no escalonador vai ser executado.

b) Posso implementar uma abordagem parecida com monitor com Events Group?

Sim, essa abordagem é sim possível de ser implementada, mas será necessário utilizar mais controladores, assim permitindo sinalizar e bloquear as atividades quando necessário.

IV. EXECUÇÃO DOS CÓDIGOS

```
1  /*
2  Arquivo: hello_word_and_blink_task.c
3  Autor: Felipe Viel
4  Função do arquivo: Cria uma task para printar o Hello World e uma para blink. Baseado no exemplo do ESP-IDF
5  Criado em 17 de novembro de 2020
6  Modificado em 17 de novembro de 2020
7  */
8
9  #include <stdio.h>
10 #include "freertos/FreeRTOS.h"
11 #include "freertos/task.h"
12 #include "esp_system.h"
13 #include "driver/gpio.h"
14 #include "nvs_flash.h"
15
16 #define BLINK_GPIO 2
17
18 > void hello_task(void *pvParameter){ ...
26
27 > void blinky(void *pvParameter){ ...
41
42 > void app_main(){ ...
```

PROBLEMAS 1 SAÍDA TERMINAL CONSOLE DE DEPURAÇÃO

Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!

powerShell
powerShell
ESP-IDF Flash Tarefa ✓
ESP-IDF Monitor

Código hello_word_and_blink_task.c

```
1  /*
2  Arquivo: task_creation_and_priority.c
3  Autor: Felipe Viel
4  Função do arquivo: Cria duas tasks para printar o Hello World e exemplifica a função para direcionar para um core. Baseado
5  Criado em 17 de novembro de 2020
6  Modificado em 17 de novembro de 2020
7  */
8
9  #include <stdio.h>
10 #include "freertos/FreeRTOS.h"
11 #include "freertos/task.h"
12 #include "esp_system.h"
13 #include "nvs_flash.h"
14 #include "esp_task.h"
15
16 > void hello_task2(void *pvParameter){ ...
22
23 > void hello_task1(void *pvParameter){ ...
29
30 > void app_main(){ ...
```

PROBLEMAS 1 SAÍDA TERMINAL CONSOLE DE DEPURAÇÃO

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

powerShell
powerShell
ESP-IDF Flash Tarefa ✓
ESP-IDF Monitor

Código task_creation_and_priority.c

```
1 > /* Touch Pad Interrupt Example...
9 > #include <stdio.h>...
18
19 static const char *TAG = "Touch pad";
20
21 #define TOUCH_THRESH_NO_USE (0)
22 #define TOUCH_THRESH_PERCENT (80)
23 #define TOUCHPAD_FILTER_TOUCH_PERIOD (10)
24
25 static bool s_pad_activated[TOUCH_PAD_MAX];
26 static uint32_t s_pad_init_val[TOUCH_PAD_MAX];
27 |
28 > /*...
37 > static void tp_example_set_thresholds(void) ...
50 > /*...
67 > static void tp_example_read_task(void *pvParameter){ ...
85 > /*...
89 > static void tp_example_rtc_intr(void *arg) ...
100 /* * Before reading touch pad, we need to initialize the RTC IO. */
101 > static void tp_example_touch_pad_init(void) ...
108
109 > void hello_world(void *pvParameter){ ...
115
116 > void app_main(void) ...
144
```

PROBLEMAS 1 SAÍDA TERMINAL CONSOLE DE DEPURACÃO

Hello World

Hello World

Hello World

powerShell

powerShell

ESP-IDF Flash Tarefa ✓

ESP-IDF Monitor

Código Touch_Pad_Interrupt_Example

```
1 #include <driver/gpio.h>
2 #include <freertos/FreeRTOS.h>
3 #include <freertos/task.h>
4 #include <freertos/semphr.h>
5 #include <freertos/event_groups.h>
6 #include <esp_system.h>
7 #include <esp_log.h>
8
9 QueueHandle_t buffer; //Cria um objeto da queue
10 //EventGroupHandle_t evt; //Cria um objeto do Grupo de Eventos
11
12 > //define EV_1SEG (1<<0) ...
16
17 > void task1(void *pvParameter){ ...
42
43 > void task2(void *pvParameter){ ...
61
62 > void app_main() ...
```

PROBLEMAS 1 SAÍDA TERMINAL CONSOLE DE DEPURACÃO

Item nao recebido, timeout expirou!

Item nao recebido, timeout expirou!

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

powerShell

powerShell

ESP-IDF Flash Tarefa ✓

ESP-IDF Monitor

```
1
2 /* FreeRTOS Real Time Stats Example
3    This example code is in the Public Domain (or CC0 licensed, at your option.)
4    Unless required by applicable law or agreed to in writing, this
5    software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
6    CONDITIONS OF ANY KIND, either express or implied.
7 */
8
9 > #include <stdio.h>...
13 |
14 #define CONFIG_LED_PIN 2
15 #define ESP_INTR_FLAG_DEFAULT 0
16 #define CONFIG_BUTTON_PIN 0
17
18 TaskHandle_t ISR = NULL;
19
20 // interrupt service routine, called when the button is pressed
21 > void IRAM_ATTR button_isr_handler(void* arg) { ...
26
27 // task that will react to button clicks
28 > void button_task(void *arg) ...
38
39 > void app_main() ...
```

PROBLEMAS 1 SAÍDA TERMINAL CONSOLE DE DEPURAÇÃO

```
I (255) heap_init: At 3FFB31B8 len 0002CE48 (179 KiB): DRAM
I (262) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (268) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (274) heap_init: At 4008AFC0 len 00015040 (84 KiB): IRAM
I (282) spi_flash: detected chip: generic
I (285) spi_flash: flash io: dio
I (290) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
```

powerShell
powerShell
ESP-IDF Flash Tarefa ✓
ESP-IDF Monitor

Código FreeRTOS_Time_Stats_Example

```
1 /*
2 Arquivo: hello_word_task.c
3 Autor: Felipe Viel
4 Função do arquivo: Cria uma task para printar o Hello World. Baseado no exemplo do ESP-IDF
5 Criado em 17 de novembro de 2020
6 Modificado em 17 de novembro de 2020
7 */
8
9 #include <stdio.h>
10 #include "freertos/FreeRTOS.h"
11 #include "freertos/task.h"
12 #include "esp_system.h"
13 #include "nvs_flash.h"
14
15 > void hello_task(void *pvParameter) ...
26
27 > void app_main() ...
```

PROBLEMAS 1 SAÍDA TERMINAL CONSOLE DE DEPURAÇÃO

```
I (0) cpu_start: Starting scheduler on APP CPU.
Hello world!
Restarting in 10 seconds...
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
Restarting in 6 seconds...
Restarting in 5 seconds...
Restarting in 4 seconds...
Restarting in 3 seconds...
Restarting in 2 seconds...
Restarting in 1 seconds...
```

powerShell
powerShell
ESP-IDF Flash Tarefa ✓
ESP-IDF Monitor

Código hello_word_task.c

```
1  /* Touch Pad Read Example
2     This example code is in the Public Domain (or CC0 licensed, at your option.)
3     Unless required by applicable law or agreed to in writing, this
4     software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
5     CONDITIONS OF ANY KIND, either express or implied.
6 */
7 #include <stdio.h>
8 #include "freertos/FreeRTOS.h"
9 #include "freertos/task.h"
10 #include "driver/touch_pad.h"
11 #include "esp_log.h"
12 #define TOUCH_PAD_NO_CHANGE    (-1)
13 #define TOUCH_THRESH_NO_USE    (0)
14 #define TOUCH_FILTER_MODE_EN    (0)
15 #define TOUCHPAD_FILTER_TOUCH_PERIOD (10)
16 /* Read values sensed at all available touch pads. Print out values in a loop on a serial monitor. */
17 > static void tp_example_read_task(void *pvParameter) ...
46
47 > static void tp_example_touch_pad_init(void) ...
53
54 > void app_main(void) ...
71
```

Código Touch_Pad_Read_Example

```
main > C hello_world_main.c > app_main(void)
1  /* Hello World Example
2
3      This example code is in the Public Domain (or CC0 licensed, at your option.)
4
5      Unless required by applicable law or agreed to in writing, this
6      software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
7      CONDITIONS OF ANY KIND, either express or implied.
8  */
9  #include <stdio.h>
10 #include "sdkconfig.h"
11 #include "freertos/FreeRTOS.h"
12 #include "freertos/task.h"
13 #include "esp_system.h"
14 #include "esp_spi_flash.h"
15
16 > void app_main(void){...
```

PROBLEMAS SAÍDA **TERMINAL** CONSOLA DE DEPURACÃO

This is esp32 chip with 2 CPU core(s), WiFi/BT/BLE, silicon revision 1, 4MB external flash
Minimum free heap size: 291384 bytes
Restarting in 10 seconds...
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
Restarting in 6 seconds...
Restarting in 5 seconds...
Restarting in 4 seconds...
Restarting in 3 seconds...
Restarting in 2 seconds...
Restarting in 1 seconds...
Restarting in 0 seconds...

Build Successfully

Código Hello_Word_Example.c