Laboratório – Gerenciamento de tarefas

Nesta seção iremos desenvolver atividades relacionadas ao uso de tarefas no FreeRTOS.

Laboratório – Aplicação baseada em tarefas

Task-01

Nessa atividade iremos desenvolver nossa primeira aplicação baseada em tarefas. Vamos escrever duas tarefas, cada uma responsável por piscar um led.

Copie o projeto “Freertos-01” e renomeie para “Task-01”

Apague o conteúdo do arquivo main.c e insira os includes dos arquivos de cabeçalho do FreeRTOS e a rotina de delay que utilizaremos nas tarefas:

```
#include "FreeRTOS.h"
#include "task.h"
#include "drivers.h"

void delay()
{
    unsigned int i;

    for(i = 0; i < 2000000; i++) {
        asm("nop");
    }
}
```

Agora insira a implementação das tarefas:

```
void taskLed1(void *pvParameters)
{
	for (;;) {
		drvLedsSet(DRV_LEDS_GREEN, DRV_LEDS_TOGGLE);
		delay();
	}
}
void taskLed2(void *pvParameters)
{
	for (;;) {
		drvLedsSet(DRV_LEDS_ORANGE, DRV_LEDS_TOGGLE);
		delay();
	}
}
```

E por último, implemente a função main():

```
int main(void)
{
    /* init CPU clock */
    drvCpuClockInit();

    /* init leds driver */
    drvLedsInit();

    /* create task 1 */
    xTaskCreate(taskLed1, (signed char *)"TaskLed1",
                configMINIMAL_STACK_SIZE, (void *)NULL, 1, NULL);

    /* create task 2 */
    xTaskCreate(taskLed2, (signed char *)"TaskLed2",
                configMINIMAL_STACK_SIZE, (void *)NULL, 1, NULL);

    /* start the scheduler */
    vTaskStartScheduler();

    /* should never reach here! */
    for(;;);
}
```

estude os parâmetros da função xTaskCreate() do freeRTOS

Compile e teste o funcionamento do FreeRTOS.

Task-02

Nesta atividade iremos aprender a compartilhar código entre as tarefas e a deletar uma tarefa em execução.

Copie o projeto “Task-01” e renomeie para “Task-02”.

Faça as seguintes alterações:

1. Como as duas tarefas tem o mesmo objetivo (piscar um led), altere-as para compartilharem o mesmo código da tarefa. Para isso, a aplicação deverá passar como parâmetro para a função xTaskCreate() o led a ser piscado.
2. Crie uma outra tarefa para piscar o led AMARELO apenas três vezes e finalizar sua execução. Neste caso, a tarefa deverá chamar no fim a função vTaskDelete(). Não esqueça também de verificar se a opção INCLUDE_vTaskDelete esta habilitada no arquivo de configuração FreeRTOSConfig.h.

Laboratório – Prioridades e escalonamento

Task-03

Nesta atividade veremos o comportamento do sistema trabalhando com tarefas em diferentes prioridades.

Copie o projeto “Task-01” e renomeie para “Task-03”.

Aumente a prioridade de uma das tarefas, compile e teste.

Task-04

Nesta atividade veremos o comportamento do sistema com o escalonador no modo colaborativo.

Copie o projeto “Task-01” e renomeie para “Task-04”.

Mude o escalonador para o modo colaborativo. Compile e teste.

Agora use a função taskYIELD() de forma que os dois leds voltem à piscar. Compile e teste.

Laboratório – Trabalhando com eventos temporais

Task-05

Nesta atividade aprenderemos a trabalhar com as rotinas de delay do FreeRTOS.

Copie o projeto “Task-02” e renomeie para “Task-05”.

Remova a tarefa 3 e a rotina de delay.

Agora implemente a tarefa que pisca o led usando a rotina de delay do FreeRTOS :

```
void taskLed(void *pvParameters)
{
    unsigned int led = (unsigned int) pvParameters;

    for (;;) {
        drvLedsSet(led, DRV_LEDS_TOGGLE);
        vTaskDelay(500/portTICK_RATE_MS);
    }
}
```

Compile e teste.

Perceba que agora as rotinas de delay são muito mais precisas que as rotinas baseadas em polling usadas anteriormente.

Task-06

Nesta atividade iremos aprender a usar a função vTaskDelayUntil() .

Copie o projeto “Task-05” e renomeie para “Task-06”.

Substitua vTaskDelay() por vTaskDelayUntil().

estude qual a diferença entre as duas chamadas, e quando você deveria usar a chamada vTaskDelayUntil().

Laboratório – Implementando a função Idle Hook

Task-07

Nesta atividade iremos medir o uso da CPU através da função Idle Hook. Mediremos a quantidade de ticks que a aplicação fica na função Idle, para então comparar com a quantidade total de ticks da aplicação, calculando a porcentagem de uso da CPU.

Copie o projeto “Task-05” e renomeie para “Task-07”.

analise a implementação da tarefa Idle do freeRTOS

Agora vamos implementar a função Idle Hook para medir o uso da CPU .

Primeiro habilite a opção configUSE_IDLE_HOOK no arquivo de configuração FreeRTOSConfig.h.

Crie uma variável global para armazenar a quantidade total de ticks e implemente a função Idle Hook no arquivo main.c.

```
static unsigned long int idle_tick_counter = 0;

void vApplicationIdleHook(void)
{
    unsigned long int tick = xTaskGetTickCount();

    while (xTaskGetTickCount() == tick);

    idle_tick_counter++;
}
```

A função Idle Hook irá contar a quantidade de ticks durante a execução da tarefa Idle e armazenar na variável idle_tick_counter.

Implemente agora a tarefa que irá calcular o uso da CPU. Os valores serão impressos na console provida pela interface de debug do Codewarrior.

```
void taskCPUUsage(void *pvParameters)
{
    unsigned long int idle_tick_last, ticks;

    idle_tick_last = idle_tick_counter = 0;

    for (;;) {
```

```
        /* wait for 3 seconds */
        vTaskDelay(3000/portTICK_RATE_MS);

        /* calculate quantity of idle ticks per second */
        if (idle_tick_counter > idle_tick_last)
            ticks = idle_tick_counter - idle_tick_last;
        else
            ticks = 0xFFFFFFFF - idle_tick_last + idle_tick_counter;
        ticks /= 3;

        /* print idle ticks per second */
        printf("%ld idle ticks per second (out of %ld)\n", ticks,
                 configTICK_RATE_HZ);

        /* calc and print CPU usage */
        ticks = (configTICK_RATE_HZ - ticks)/10;
        printf("CPU usage: %d%%\n", ticks);

        /* update idle ticks */
        idle_tick_last = idle_tick_counter;
    }
}
```

E crie esta tarefa com a menor prioridade possível:

```
/* create task to show CPU usage */
xTaskCreate(taskCPUUsage, (signed char *)"Task CPU Usage",
            configMINIMAL_STACK_SIZE * 4, (void *)NULL, 0, NULL);
```

O driver que estamos usando esta direcionando a saída da função printf() para a porta serial. Por este motivo, inicie a porta serial no início da função main(), logo depois de inicializar os leds:

```
/* init uart driver */
drvUartInit(DRV_UART_CH_DEF);
```

Compile e teste. Para testar, use um cabo serial ou conversor USB/serial para ligar o kit à sua máquina de desenvolvimento (a configuração padrão da porta serial é 115200,8N1).

Após os testes, perceba que, como as tarefas que acendem o led consomem quase nada de processamento, o uso de CPU vai aparecer como 0%. Vamos agora criar uma outra tarefa apenas para consumir CPU:

```
void taskUseCPU(void *pvParameters)
{
    unsigned int i, j;

    for (;;) {

        for (i = 0, j = 0; i < 10000; i++){
            j *= i + 12.34;
        }
```

```
        vTaskDelay(100/portTICK_RATE_MS);
    }
}
```

Crie esta tarefa na função main():

```
/* create task to use CPU */
xTaskCreate(taskUseCPU, (signed char *)"TaskUseCPU",
            configMINIMAL_STACK_SIZE, (void *)NULL, 1, NULL);
```

Agora teste novamente. O uso da CPU deve ficar entre 15% e 20%.

Perceba como medir o consumo de CPU é uma técnica importante durante o desenvolvimento de uma aplicação baseada em um RTOS. Mas a técnica que estudamos aqui é bastante imprecisa, já que sua granularidade é de 1 tick (1ms no nosso caso).

No decorrer do treinamento estudaremos uma funcionalidade disponibilizada pelo FreeRTOS para colher estatísticas de execução das tarefas da aplicação chamada Run Time Statistics, que possibilita uma precisão muito maior comparada à técnica utilizada nesta atividade.