João Génio, 88771
Ruben Menino, 89185

Docente: Paulo Pedreiras

Task Management framework for FreeRTOS

Introdução

Neste projeto foi-nos proposto criar uma framework (Task Manager - TMan) que consiga suplementar a básica funcionalidade do FreeRTOS e permita o registo e a ativação de várias tarefas criadas de uma forma coordenada, conseguindo especificar explicitamente atributos como o período, fases relativas, precedências entre várias das tarefas criadas e conseguir ainda ativar essas tarefas em instantes apropriados.

Funcionalidades implementadas

No desenvolvimento deste projeto, foram realizados incrementalmente vários passos para chegar a uma framework funcional.

Inicialmente, e para conseguir posteriormente implementar outras das funcionalidades pretendidas, foi implementado o suporte básico para a ativação das tarefas periodicamente. Após obtermos esta ativação periódica das tarefas funcional, conseguimos então adicionar certos atributos a partir do método TMAN_TaskRegisterAttributes às tarefas adicionadas anteriormente.

Foi implementada então, a possibilidade de definir uma fase para uma determinada tarefa criada. Se a fase não for atribuída, o tempo de lançamento dessa tarefa é considerado zero. Ao definir por exemplo uma fase 1 a uma tarefa, essa mesmo é lançada em t = 1.

Foi ainda implementada a possibilidade de definir se alguma tarefa criada teria de depender da execução da outra. No caso de existirem três tarefas (tarefa 1, tarefa 2, tarefa 3) é possível fazer uma simples precedência, atribuindo por exemplo que a tarefa 2 é dependente da tarefa 1.

Estrutura de dados utilizada

Para ser mais cómodo e mais fácil de utilizar, foi criada uma struct com o nome “taskInfo”, onde guarda as informações das tarefas que vão ser criadas. Esta struct define tipos de dados que agrupam variáveis sob um mesmo tipo de dado. Com

este struct permite-nos que, ao armazenar os dados de uma mesma entidade, isto possa ser feito com uma única variável.

Nesta struct, temos então relativamente a cada tarefa, o “id” correspondente, que vai ser o principal identificador da tarefa, o “period”, que vai definir o intervalo de tempo em que a tarefa se repete, a “phase”, a “precendence”, o “nActivations” que guarda o número de vezes que a tarefa foi ativada, o “nCompletions” que guarda o número de vezes que o código da tarefa foi executada e finalmente o handle do semáforo, para conseguir criar um semáforo a partir do método xSemaphoreCreateBinary() para cada uma das tarefas adicionadas, e posteriormente conseguir dar give e take desses semáforos. Foi criado também um array de 6 posições para as 6 respectivas tarefas criadas, onde foi utilizada essa struct e foram guardados os valores das tarefas.

Descrição da API implementada, incluindo os argumentos dos métodos

- **TMAN_Init(int tick)**
    - Este método é onde é inicializada a framework. Aqui é criada uma tarefa com o nome “tick_updater” que vai agir como tarefa principal, onde vai gerir todas as tarefas adicionadas à framework. Como era necessário a periodicidade, a phase e outros atributos serem um número inteiro de TMAN ticks, foi necessário converter os ticks do freeRTOS para ticks da framework TMAN. O argumento tick é definido no início do programa.

- **TMAN_Close()**
    - É utilizado o método vTaskDelete(updateHandle) com o handle da tarefa principal. Após o scheduling a tarefa é removida do gerenciamento dos kernels RTOS.

- **TMAN_TaskAdd()**
    - Neste método, é criada uma estrutura com várias informações das tarefas e a inicialização de alguns valores, como o número de ativações e de conclusões.

- **TMAN_RegisterAtributes(int task_id, int period, int deadline, int phase, int precedence)**
    - Neste método é então criado um array do respetivo id da tarefa que é passada como argumento da função, onde são guardadas informações da tarefa.
    - Vai permitir posteriormente definir os valores do período, da fase e da precedência da respectiva tarefa.

- **TMAN_TaskWaitPeriod(int task_id)**
    - Esta função faz parte do sistema de gestão de tarefas. No início do código de cada tarefa, esta função é chamada para bloquear a tarefa em questão, usando o mecanismo “Take” do seu semáforo a esta associado. Isto significa que uma tarefa é bloqueada sempre que inicia a execução do seu código. Esta função incrementa o número de vezes que a tarefa executou o seu código (inicializado a -1, porque esta função é chamada antes da primeira execução do código da tarefa)

tick_updater(void *pvParam)

- Esta é a tarefa que é lançada na inicialização da framework. O seu trabalho é:
    - converter “ticks” FreeRTOS para “ticks” TMAN, explicado anteriormente.
    - percorrer todas as estruturas das tarefas adicionadas à framework, e decidir quando podem ser “desbloqueadas” (através do “Give” do seu semáforo)

Precedência

- A nossa implementação de precedência, apenas verifica se a tarefa que tem de executar antes da que estamos a iterar, já completou mais execuções que a atual. Isto obriga a que uma tarefa A, que tem de executar antes de uma tarefa B, execute 1 vez para B poder executar 1 vez também.

Fase e Período

- Para desbloquearmos uma tarefa pela primeira vez, verificamos se o número de “ticks” TMAN passados, é igual à sua fase.
- Para a tarefa executar no seu período, tem de ser desbloqueada num “tick” TMAN múltiplo do seu período. Para termos em conta a fase neste cálculo, usamos a fórmula:

```
(tick_now - task_list[i].phase) %
task_list[i].period == 0
```

- **TMAN_TaskStats(int task_id)**
    - Este método é chamado na função TMAN_TaskWaitPeriod, e após ser efetuado o take do semáforo da tarefa correspondente, e após incrementar o número de conclusões da tarefa, é então chamado este método, caso o número de ativações dessa respetiva tarefa seja igual a 3. São imprimidas então as informações acerca do número de

ativações e do número de conclusões do respetivo id da tarefa que é passado como argumento da função.

Descrição dos testes realizados e resultados obtidos

```
TMAN_TaskAdd();
TMAN_TaskRegisterAtributes(idA, 1, 1, 0, -1); // id, period, deadline, phase, precedence
xTaskCreate( simpleA, "A", configMINIMAL_STACK_SIZE, NULL, PRIO_A, NULL );

TMAN_TaskAdd();
TMAN_TaskRegisterAtributes(idB, 1, 1, 0, -1);
xTaskCreate( simpleB, "B", configMINIMAL_STACK_SIZE, NULL, PRIO_B, NULL );

TMAN_TaskAdd();
TMAN_TaskRegisterAtributes(idC, 2, 1, 0, 4);
xTaskCreate( simpleC, "C", configMINIMAL_STACK_SIZE, NULL, PRIO_C, NULL );

TMAN_TaskAdd();
TMAN_TaskRegisterAtributes(idD, 2, 1, 1, -1);
xTaskCreate( simpleD, "D", configMINIMAL_STACK_SIZE, NULL, PRIO_D, NULL );

TMAN_TaskAdd();
TMAN_TaskRegisterAtributes(idE, 5, 1, 2, -1);
xTaskCreate( simpleE, "E", configMINIMAL_STACK_SIZE, NULL, PRIO_E, NULL );

TMAN_TaskAdd();
TMAN_TaskRegisterAtributes(idF, 10, 1, 0, -1);
xTaskCreate( simpleF, "F", configMINIMAL_STACK_SIZE, NULL, PRIO_F, NULL );
```

```
A, 0
B, 0
F, 0
A, 1
B, 1
D, 1
A, 2
B, 2
E, 2
A, 3
B, 3
D, 3
A, 4
B, 4
C, 4
A, 5
B, 5
D, 5
```

```
A, 6
B, 6
A, 7
B, 7
D, 7
E, 7
A, 8
B, 8
C, 8
A, 9
B, 9
D, 9
A, 10
B, 10
F, 10
```

```
A, 11
B, 11
D, 11
A, 12
B, 12
E, 12
A, 13
B, 13
D, 13
A, 14
B, 14
C, 14
```

Neste respetivo caso foram então criadas 6 tarefas diferentes, com 3 prioridades distintas, e com os seguintes atributos :

A = tskIDLE_PRIORITY + 3 | period = 1 | phase = 0 | precedence = -1
B = tskIDLE_PRIORITY + 3 | period = 1 | phase = 0 | precedence = -1
C = tskIDLE_PRIORITY + 2 | period = 2 | phase = 0 | precedence = 4
D = tskIDLE_PRIORITY + 2 | period = 2 | phase = 1 | precedence = -1
E = tskIDLE_PRIORITY + 1 | period = 5 | phase = 2 | precedence = -1
F = tskIDLE_PRIORITY + 0 | period = 10 | phase = 0 | precedence = -1

Adicionando estas tarefas é possível testar todas as funcionalidades implementadas como atribuição de um período, de uma phase e de casos de precedência.

Primeiramente, iniciam as tarefas com phase 0, neste caso iniciou A, B e F(ordem de prioridade).
A C mesmo tendo phase 0, não iniciou pois tem precedência da tarefa E, e como E tem uma phase de 2, ainda não executou. Após E executar, C executa.