SOTR Report

Martim Neves, 88904
Daniel Correia, 90480

February 8, 2022

1 Funcionalidades Implementadas

As funcionalidades implementadas foram todas as funções obrigatórias (Init, Close, TaskAdd, TaskRegisterAttributes, TaskWaitPeriod e TaskStats) e, adicionalmente, foram implementadas as funcionalidades de modificação dinâmica dos atributos das tarefas (período, fase, *deadline* e precedências) e também a deteção de *deadline misses*, sendo que também oferecemos a possibilidade de o utilizador poder escrever uma função de *callback* que é chamada quando acontece uma *deadline miss*.

2 Estruturas de Dados Utilizadas

Foi necessário criar uma *struct* denominada de **Task** onde são guardadas todas as informações necessárias de uma tarefa adicionada à framework. É declarado um array estático com dimensão *TMAN_MAX_TASKS* (uma macro definida no ficheiro tman.h) com dimensão que pode ser modificada pelo utilizador, onde são guardadas as várias **Task** adicionadas à framework. Em baixo encontra-se uma sucinta descrição da *struct* **Task** criada:

- **pcName** - o nome que o utilizador deu a uma tarefa
- **taskPeriod** - o período da tarefa em TMAN ticks, caso esta variável seja zero é considerada como uma tarefa esporádica
- **taskPhase** - a *phase/initial offset* da tarefa em TMAN ticks
- **taskDeadline** - a deadline relativa da tarefa em TMAN ticks
- **taskPrecedenceConstrains** - caso a tarefa tenha precedências, é o nome da tarefa que tem como precedente. Se não houver tarefas precedentes é apenas um ponteiro para uma string vazia
- **taskActivations** - um contador sobre o número de activações da tarefa. É incrementado no fim da função *TaskWaitPeriod*, quando a tarefa deixa de estar bloqueada e passa a estar em execução
- **lastActivationTick** - o TMAN tick em que a tarefa foi pela última vez activada pelo *Tick Handler* ou, caso seja uma tarefa esporádica, a última vez que a tarefa precedente a activou.
- **deadlineMisses** - um contador sobre o número de deadlines falhadas pela tarefa. É incrementado quando é detectada uma deadline falhada no fim da execução da tarefa

- **xSemaphoreHandler** - um semáforo onde esta tarefa se vai bloquear durante um tempo ilimitado. Apenas irá sair deste semáforo quando o *Tick Handler* decidir que está num tick onde esta tarefa deve activar fazendo um *xSemaphoreGive*
- **xSemaphore** - um semáforo onde as tarefas que têm esta tarefa como precedente se vão bloquear durante um tempo ilimitado. Essas tarefas apenas iram sair deste semáforo quando esta tarefa fizer um *xSemaphoreGive* no fim da sua execução
- **taskPrecedenceIndex** - o index, no array estático de **Task**, da tarefa que tem como precedente
- **execute** - se o *Tick Handler* tentasse acordar uma tarefa mais que uma vez antes de ela executar (tivesse passado mais que um período da tarefa), o **lastActivationTick** iria ser atualizado apesar da tarefa ainda não ter executado. Quando a flag está a 0 o valor do **lastActivationTick** pode ser actualizado e quando a 1 o valor não pode ser actualizado até a tarefa executar e voltar a meter a flag a zero

3 Descrição dos Métodos Implementados

É de notar que grande parte dos métodos implementados retornam um código de erro para que o utilizador possa saber se correu tudo conforme esperado, ou, caso contrário, o que levou a esse erro.

3.1 Init

Esta função serve para inicializar a framework e recebe como argumentos o número de TMAN_TICKS, uma flag para indicar se o utilizador quer ou não ver as estatísticas das tarefas e um ponteiro para uma função que será chamada quando ocorre uma *deadline miss*. Inicialmente é verificada a validade dos parâmetros de entrada e, caso sejam todos válidos, é criada uma tarefa reguladora do sistema referida como **Tick Handler**. O TMAN_TICK tem que ser maior que 0 e múltiplo do tick do FreeRTOS, ambos definidos em milissegundos. Já a flag tem que ter os valores 1 ou 0, caso o utilizador queira ver as estatísticas ou não, respectivamente, e o ponteiro para a função, caso seja diferente de NULL, vai executar o código presente na região de memória indicada pelo ponteiro, caso este seja NULL, é executada a função **DeadlineCallback** criada por nós, que apenas imprime no terminal uma mensagem a dizer que determinada tarefa falhou uma *deadline*. Para testar a função de callback basta utilizar a variável criada por nós para exemplificar a funcionalidade, mudando o ponteiro de NULL para prt nos argumentos da função **Init** na main. No entanto, caso o utilizador decida fazer uma função, esta não pode ter argumentos.

3.2 Close

Nesta função são apagados o *TickHandler* e os semáforos usados de modo a libertar memória alocada no sistema.

3.3 TaskAdd

Nesta função são adicionadas as tarefas à framework, sendo que o único parâmetro de entrada é o nome a ser dado à tarefa. Caso já exista uma tarefa com o mesmo nome, é devolvido um erro. Caso o nome seja válido, a tarefa é adicionada ao array de tarefas e são criados dois semáforos para a tarefa, um para o **TickHandler** acordar as tarefas no período e fase correctos e outro para garantir o cumprimento de precedências.

3.4 TaskRegisterAttributes

Esta função recebe como argumentos o nome da tarefa, o período, a fase, a *deadline* e precedências. Caso todos os atributos sejam válidos, são associados à tarefa correspondente para posteriormente serem usados pela *framework*. Esta função permite que a precedência seja uma *string* vazia, que significa que não existe precedência. Permite também que uma tarefa seja esporádica, definindo o período como 0, e que as tarefas não tenham *deadline*, caso esta seja definida como 0.

3.5 TaskWaitPeriod

Esta função é chamada em todas as execuções de uma tarefa, sendo que o único parâmetro de entrada é o nome da tarefa que chamou esta função. Primeiramente é feita a verificação se o nome fornecido como parâmetro é válido, ou seja, se é o nome de uma tarefa previamente adicionada à framework. De seguida é feita a verificação de *deadline misses* verificando se o **lastActivationTick** + **taskDeadline** é menor que o **TMAN tick** actual, se essa verificação for verdade é porque ocorreu uma *deadline miss*. São imprimidas as estatísticas da tarefa caso o argumento da função **Init** tenha sido 1 e a tarefa faz um give ao semáforo das precedências caso não seja o primeiro **TMAN tick**. Existem então duas zonas em que as tarefas se podem bloquear. Caso não seja uma tarefa esporádica, ela vai-se bloquear no semáforo da própria tarefa, durante tempo ilimitado, até ser acordada pelo **Tick Handler**. Na segunda, as tarefas que tenham precedências vão se bloquear no semáforo da tarefa que tenham como precedente. A variável **lastActivationTick** é actualizada antes desse bloqueio caso seja uma tarefa normal, e é actualizada depois caso seja uma tarefa esporádica.
Por fim é incrementado o número de **taskActivations** e é dado o reset à flag **execute** para que a variável **lastActivationTick** possa ser actualizada.

3.6 TaskStats

Esta função, caso a flag passada à função *Init* esteja a 1, imprime no terminal o número de activações e *deadline misses* da tarefa cujo nome é passado como argumento após cada activação da tarefa.

3.7 TickHandler

Como já foi referido, houve uma necessidade de criar uma tarefa reguladora do sistema referida como **Tick Handler**, que é a tarefa com maior prioridade do sistema, e tem como função regular as activações das várias tarefas da framework. Esta é a função que a tarefa reguladora executa e foi tido em consideração que teria de ser rápida e eficiente.O **Tick Handler** começa então por se bloquear durante 1 **TMAN tick** para que as tarefas de menor prioridade (as tarefas adicionadas à framework) possam executar e bloquearem-se a si mesmas. É devido a isto que as tarefas adicionadas à framework não começam a executar logo no **FreeRTOS tick** 0, e sim 1 **TMAN tick** equivalente a **FreeRTOS ticks** a seguir.

Quando o **Tick Handler** volta do estado bloqueado (1 **TMAN tick** depois) ele vai percorrer todas as tarefas adicionadas à framework e faz as seguintes verificações apenas para tarefas não esporádicas: se o TMAN tick actual for igual ao período da tarefa ele acorda a mesma e actualiza o **lastActivationTick** para o tick actual. Se o TMAN tick actual for maior que a fase da tarefa e se o resto da divisão de (TMAN tick actual - fase da tarefa) por período da tarefa for igual a 0, o **Tick Handler** vai acordar a tarefa e actualizar o **lastActivationTick** para o tick actual. Por fim vai incrementar o TMAN Tick actual.

3.8 DeadlineCallback

Esta função é chamada quando ocorre uma *deadline miss* e imprime uma mensagem a dizer que uma certa tarefa, cujo nome é passado como argumento, falhou uma *deadline*.

4 Testes e Resultados

Para testar as funcionalidades obrigatórias começámos por fazer testes simples, de modo a garantir que tudo estava a funcionar correctamente, e fomos modificando a configuração de modo a testar casos mais complexos e edge cases, para verificar o correto funcionamento nesses casos. De modo a testar as *deadline misses* criámos apenas duas tarefas, a primeira com período 3, fase 0, sem *deadline* e sem precedências, e a segunda com período 1, fase 0, *deadline* 2 e com precedência da primeira tarefa, de modo a forçar que a segunda tarefa falhe a *deadline*. O resultado é mostrado na figura 1.

```
 ******************************************
Starting TMAN FreeRTOS Demo - Simple Test TMAN Framework
******************************************
A, 200
B, 200
A, 800
B, 800
A, 1400
B, 1400
Task B missed a deadline!
A, 2000
B, 2000
Task B missed a deadline!
```

Figure 1: *Deadline miss*

Para testar a mudança dinâmica de atributos criámos duas tarefas novamente, a primeira com período 2, fase 0, sem *deadline* e sem precedências, e a segunda com período 3, fase 0, sem *deadline* e sem precedências. A dada altura, os atributos da segunda tarefa são alterados (quando aparece a mensagem "aqui") para período 4, fase 1, sem *deadline* e com precedência da primeira tarefa. De acordo com os nossos testes, todos os atributos estão a ser alterados correctamente, no entanto, não é possível fazer com que uma tarefa periódica passe a ser esporádica. O resultado pode ser visto na figura 2.

```
 ******************************************
Starting TMAN FreeRTOS Demo - Simple Test TMAN Framework
******************************************
A, 200
B, 200
A, 600
B, 800
A, 1000
A, 1400
B, 1400
A, 1800
B, 2000
A, 2200
aqui
A, 2600
B, 2800
A, 3000
A, 3400
B, 3600
```

Figure 2: Mudança dinâmica dos atributos