

SOTR - Task Management framework for FreeRTOS

Rodrigo Lopes da Silva Santos, nº mec 89180

Rui Pedro Pereira Santos, nº mec 89293

### **Funcionalidades implementadas:**

As funcionalidades implementadas neste projeto consistem no suporte básico de ativação de uma tarefa periódica, com a possibilidade de especificar o período de uma tarefa em ticks do sistema de gestão de tarefas TMAN implementado, a possibilidade de definir uma fase às tarefas também especificada em ticks do sistema, a capacidade de atribuir um deadline relativo e também a possibilidade de definir restrições de precedência entre tarefas.

Para além destas funcionalidades obrigatórias foram também implementadas como extra a permissão de atribuição de precedência múltipla, ou seja, uma tarefa depender de uma ou mais tarefas, bem como a deteção de deadline misses e a possibilidade de modificar o período e a fase das tarefas.

### **Estruturas de dados utilizados:**

Para fazer a sua gestão, foi criada uma estrutura de dados para a definição dos vários parâmetros das tarefas.

A estrutura apresenta o seguinte formato:

```
char name
int priority
int period
int activations
int phase
int deadline
int deadline_misses
int ready
int precedence[5]
TaskHandle_t handler
```

A estrutura possui atributos para armazenar o nome, prioridade, período, número de ativações, fase, deadline, deadline misses, assim como um atributo que serve de flag para assinalar quando a tarefa se encontra pronta a executar. Contém um array de identificadores para armazenar as precedências da tarefa bem como um identificador *TaskHandle\_t* usado quando se pretende fazer o seu *resume*.

### **Descrição da API implementada:**

A API implementada conta com um conjunto de 6 funções, *TMAN\_Init*, *TMAN\_Close*, *TMAN\_TaskAdd*, *TMAN\_TaskRegisterAttributes*, *TMAN\_TaskWaitPeriod*, *TMAN\_TaskStats*, que se encontram detalhadas a seguir.

*TMAN\_Init* ( int TMAN\_TICK\_PERIOD\_VALUE, int N\_TASKS ):

Função de inicialização da framework, onde é alocada a memória necessária para armazenar o array de estruturas de tarefas com tamanho *N\_TASKS* ( número de tarefas a gerir ). É nesta função que é lançada a tarefa responsável pelo TICK do sistema, onde

*TMAN\_TICK\_PERIOD* é o número de ticks do FreeRTOS que equivalem a um TICK do nosso Task Manager.

*TMAN\_TaskAdd* ( char name ):

Método responsável por inicializar as flags e nome de uma tarefa, bem como chamar a função *xTaskCreate()* para criação e lançamento da designada tarefa. Após a criação da tarefa, esta é colocada em suspensão imediatamente.

*TMAN\_TaskRegisterAttributes* ( char name, int period, int phase, int deadline, int precedence\_constraints[ ] ):

Função de registo dos atributos de uma tarefa já adicionada. A tarefa em questão é obtida através do seu nome, e os atributos período, fase, deadline, e *precedence\_constraints[ ]* são inicializados.

*TMAN\_TaskWaitPeriod* ( void ):

Função chamada pelas tarefas quando terminam a sua execução. Nesta função é executado o método *vTaskSuspend()* com o argumento *NULL* para que a tarefa se auto-suspenda.

*TMAN\_TaskStats* ( void ):

Função que imprime informação estatística relativa a cada tarefa, nomeadamente o seu número de ativações e o número de *deadline misses*.

*TMAN\_Close*( void ):

Função que termina a execução da framework. Para cada tarefa criada, executa a função *vTaskDelete()* e por fim a função *vTaskEndScheduler()*, que pára o *TICK* do freeRTOS.

Para além destas funções de interação com a plataforma também foram criados mecanismos para o funcionamento da mesma. Nisto inclui-se uma tarefa mais prioritária responsável pelo *TICK* do sistema. Esta tarefa chama a função *TMAN\_TaskStats* a cada ciclo, incrementa o *TICK* do sistema ( *TMAN\_TICK* ), chama a função responsável pelo tratamento das tarefas ( *task\_manager*( ) ) e faz a chamada ao método *vTaskDelayUntil()* do freeRTOS, colocando-se em bloqueio *TMAN\_TICK\_PERIOD* número de ciclos do freeRTOS.

A função *task\_manager()* é responsável pelo *scheduling*, ou seja, faz a gestão das tarefas e quando devem ser executadas.

Para isso, foi criado um *for loop* onde todas as tarefas são percorridas e é verificado se o resto da divisão inteira entre o *TMAN\_TICK* e o período de cada tarefa é igual à sua fase.

Nesse caso, os atributos *ready* e *activations* da tarefa são incrementados, o que significa que se pretende executar um *resume* desta tarefa.

Após ter percorrido todas as tarefas, o *scheduler* já determinou quais é que ficaram *ready* e pode agora, através de outro ciclo, executar o *resume* das mesmas tendo em conta as condições de precedência.

De seguida, são novamente percorridas todas as tarefas e é verificado se estas têm o seu atributo *ready* superior a 0 e é inicializada uma variável *dont\_execute* a 0. Nesse caso, é percorrido o seu atributo *precedence* (array com os índices das tarefas com relação de precedência) e é verificado se estas têm o seu atributo *ready* superior a 0. Nesse caso, a flag *dont\_execute* é incrementada, o que significa que a tarefa atual não deve correr neste ciclo visto que tem relações de precedência com outras tarefas que também devem ser

executadas. Após percorrer o atributo *precedence* é verificado se a flag *dont\_execute* é igual a 0 e, nesse caso, é executado o *vTaskResume(TASKS[task].handler)* da tarefa. Ao executar, a tarefa deve imprimir o seu nome e tick em que está a executar, deve decrementar o seu atributo *ready* e por fim deve executar a função *TMAN\_TaskWaitPeriod*. Por fim, para verificar se ocorreu um *deadline miss*, é verificado se o resto da divisão inteira entre o *TMAN\_TICK* e o período da tarefa é igual à soma entre o *deadline* e a fase da tarefa. Neste caso, é imprimido uma frase assinalando que houve um *deadline miss* e o atributo *deadline\_misses* da tarefa é incrementado.

### Testes e resultados obtidos:

Ao longo da sua implementação, a plataforma foi testada para verificar o seu correto funcionamento. Para facilitar os testes e a percepção, cada tarefa ao executar, imprime o seu nome e o tick em que executou. ( ex: "A, 20" )

COM3 - PuTTY

```
*****
TMAN - Task Manager framework for FreeRTOS
*****
F, 3
C, 3
A, 5
D, 6
F, 6
C, 6
E, 7
F, 9
C, 9
B, 10
A, 10
D, 12
F, 12
C, 12
E, 14
F, 15
C, 15
A, 15
D, 18
F, 18
C, 18
B, 20
A, 20
```

Nesta execução da framework foram criadas 6 tarefas com períodos diferentes, A - 5, B - 10, C - 3, D - 6, E - 7, F - 3. As tarefas tinham as seguintes prioridades: A - 0, B - 1, C - 2, D - 3, E - 3, F - 3. Como podemos verificar na imagem 1 tanto a prioridade como a periodicidade se encontram a funcionar corretamente, visto que Tarefas com períodos iguais, como no caso da tarefa C e F, são executadas por ordem de prioridade, sendo que a F imprime sempre primeiro que a C.

Figura 1: Teste de periodicidade e prioridade

COM3 - PuTTY

```
*****
TMAN - Task Manager framework for FreeRTOS
*****
D, 1
E, 1
B, 1
A, 2
F, 3
C, 3
F, 6
C, 6
D, 7
A, 7
E, 8
F, 9
C, 9
B, 11
F, 12
C, 12
A, 12
D, 13
E, 15
F, 15
C, 15
A, 17
F, 18
C, 18
D, 19
```

Com as mesmas configurações anteriores foi adicionada uma fase a algumas tarefas: A - 2, B - 1, D - 1, E - 1. E como é possível verificar na figura 2, a tarefa A por exemplo corre em *TICKs* iguais a período + fase (7, 12 e 17).

Figura 2: Teste de fase

```

COM3 - PuTTY
*****
TMAN - Task Manager framework for FreeRTOS
*****
D, 1
E, 1
B, 1
A, 2
F, 3
C, 4
F, 6
D, 7
C, 7
A, 7
E, 8
F, 9
C, 10
B, 11
F, 12
A, 12
D, 13
C, 13
E, 15
F, 15
C, 16
A, 17
F, 18
D, 19
C, 19
F, 21
E, 22
C, 22
A, 22
----- TASK (B) DEADLINE MISS!
B, 23

```

Continuando com as configurações anteriores, foram adicionadas as seguintes precedências simples: B depende de C e C depende de F. Também foram adicionados deadlines por exemplo a tarefa B tem um deadline de 2 *TICKS*. Como é possível ver na figura 3, quando a tarefa C e F querem executar no mesmo *TICK*, a tarefa C espera pela execução da tarefa F, e apenas executa no *TICK* a seguir. ( F, 3 -> C, 4 ). No *TICK* 21 as tarefas B, C e F pretendem executar devido aos períodos e fases. Assim, visto que a tarefa B apenas tem um deadline de 2 e depende da tarefa C que por sua vez depende da tarefa F, apenas executa dois *TICKS* depois da sua ativação, o que resulta num deadline miss.

**Figura 3:** Teste precedência simples e deadline

```

----- TASK (B) DEADLINE MISS!
B, 23
TASK (A) NUMBER OF ACTIVATIONS = (5)
TASK (A) DEADLINE MISSES = (0)
TASK (B) NUMBER OF ACTIVATIONS = (3)
TASK (B) DEADLINE MISSES = (1)
TASK (C) NUMBER OF ACTIVATIONS = (7)
TASK (C) DEADLINE MISSES = (0)
TASK (D) NUMBER OF ACTIVATIONS = (4)
TASK (D) DEADLINE MISSES = (0)
TASK (E) NUMBER OF ACTIVATIONS = (4)
TASK (E) DEADLINE MISSES = (0)
TASK (F) NUMBER OF ACTIVATIONS = (7)
TASK (F) DEADLINE MISSES = (0)

```

Seguindo o teste anterior, para garantir que a contagem de ativações e deadline misses se encontra correta, foi impresso as estatísticas das tarefas com recurso à função `TMAN_TaskStats()`, e podemos ver que após o deadline miss este é apresentado nas estatísticas da tarefa B, bem como o número de ativações de cada uma das tarefas.

**Figura 4:** Teste à contabilidade de estatísticas

```

COM3 - PuTTY
*****
TMAN - Task Manager framework for FreeRTOS
*****
D, 1
E, 1
A, 2
F, 3
C, 4
F, 6
D, 7
C, 7
A, 7
E, 8
F, 9
B, 10
C, 11

```

Para testar a precedência múltipla, colocamos a fase da tarefa B a 0, a tarefa C depende agora da tarefa B e F e a tarefa B não tem dependências. De acordo com o esperado, a tarefa C que normalmente executaria no *TICK* 9, executa apenas 2 *TICKS* depois visto que depende da tarefa F que executa no *TICK* 9 e da tarefa B que executa no *TICK* 10.

**Figura 5:** Teste precedência múltipla