

## SOTR 21/22 – Final project

### Task Management framework for FreeRTOS

Pedro Almeida (89205), Renato Valente (89077)

#### Introdução

O desafio proposto consiste em desenvolver uma framework (Task Manager - TMan) que permite registar um conjunto de tarefas FreeRTOS e associado a cada uma delas um conjunto de atributos como o período, deadline e fase e ativar essas tarefas mesmo nos momentos apropriados.

As funcionalidades implementadas foram as seguintes: suporte para ativação de tarefas periódicas, possibilidade de atribuir uma fase relativa a uma tarefa, possibilidade de atribuir um prazo relativo a uma tarefa e possibilidade de definir restrições de precedência.

#### Estrutura de dados

Um dos primeiros passos no desenvolvimento da framework TMan foi a definição de uma estrutura de dados que representasse uma tarefa. Ao longo do desenvolvimento foram sendo adicionados atributos necessários para a implementação de funcionalidades mais avançadas. Por fim, uma *Task* tem os seguintes atributos:

- **name** - nome da tarefa
- **deadline** - número do TMan Tick que corresponde à deadline da tarefa
- **phase** - número do TMan Tick da fase da tarefa
- **period** - número do TMan Tick
- **precedence**
  - nome da tarefa com precedência
  - a precedência das tarefas periódicas (sem precedência) é “x”
- **currentActivation** - número do TMan Tick em que a tarefa foi ativada
- **nextActivation** - número do TMan Tick em que a tarefa vai ser ativada
- **numberOfActivations**
  - contador do número de ativações
  - usado para estatística
- **deadlineMissedCounter**
  - contador do número de falhas de deadline
  - usado para estatística
- **state** - três possíveis estados:
  - Started - tarefa com os atributos registados; ainda não houve nenhuma execução
  - Running - tarefa encontra-se em execução
  - Blocked - tarefa está bloqueada à espera da próxima ativação
- **end** - número do TMan Tick em que a tarefa acabou de executar

#### Implementação

A framework desenvolvida suporta dois tipos de tarefas: periódicas e esporádicas. Uma tarefa torna-se esporádica caso tenha alguma precedência.

Para gerir as tarefas foi criado um array de tamanho estático mas configurável antes da utilização da framework.

A API desenvolvida tem os seguintes métodos:

### **TMan\_Init (int nMax)**

Este método é responsável por inicializar a *framework* TMan. É feita a inicialização de algumas variáveis necessárias para a utilização da API, como o número de tarefas adicionadas (zero), o número máximo de tarefas permitido (passado por argumento) e uma fila responsável por armazenar mensagens a imprimir. É também feita a criação de duas tarefas necessárias para a utilização da API. Uma dessas tarefas é a “ticks”, com prioridade mais elevada que todas as outras tarefas e responsável por contar os ticks do sistema (**TMan\_Tick**) e ativar e desativar as restantes tarefas. É esta tarefa que decide o fluxo de execução das outras tarefas. A outra tarefa criada inicialmente é a tarefa “prints”, com a prioridade mais baixa que todas as outras tarefas e é responsável por imprimir as mensagens armazenadas na fila (**QueueHandle\_t**) de forma ordenada.

Assim, as prioridades foram definidas da seguinte maneira:

PRIOTIRY\_TICKS > PRIORITY\_TASK > MSG\_PRIOTIRY

### **TMan\_TaskAdd (const char\* taskName)**

Este método é responsável por adicionar uma tarefa ao *array* de tarefas. Recebe como argumento o nome da tarefa a adicionar. Foi também implementada lógica adicional para verificar se o número de tarefas adicionadas é menor que o máximo permitido e se não existe uma outra tarefa com o mesmo nome. Só depois destas verificações é que de facto é feita a adição da tarefa ao *array*.

### **TMan\_TaskRegisterAttributes (int index, int phase, int period, int deadline)**

Neste método registam-se os atributos de uma tarefa periódica. Todos os atributos de uma *Task* são inicializados aqui. A tarefa é identificada pelo index (passado por argumento) no array de tarefas adicionadas. Os atributos phase, period e deadline são também passados por argumento. Os restantes atributos são declarados com os seus valores iniciais, isto é, os contadores de número de ativações e falhas de deadline são iniciados a zero, o state a *started*, currentActivation e nextActivation são iguais à fase e, sendo uma tarefa periódica, não tem precedência.

### **TMan\_SporadicTaskRegisterAttributes (int index, int deadline, const char\* precedence)**

Neste método registam-se os atributos de uma tarefa esporádica. Uma tarefa torna-se esporádica caso tenha alguma precedência. À semelhança do método anterior, a tarefa é identificada pelo index no array de tarefas adicionadas. A deadline e a precedência são passadas por argumento. Os restantes atributos são todos iniciados a zero.

### **TMan\_TaskWaitPeriod** - a task espera pela próxima ativação, verifica falha da deadline

Nesta função, a tarefa encontra-se bloqueada (vTaskSuspend) à espera da próxima ativação. Enquanto a espera decorre, é feita uma verificação de falha de deadline. Caso o valor do TMan Tick seja superior à currentActivation mais a deadline significa que a tarefa falhou a deadline ainda antes de ter sido ativada.

### **TMan\_TaskStats (const char\* taskName)**

Este método apenas constrói uma mensagem com as estatísticas da tarefa passada por argumento. As estatísticas são o número de ativações e o número de deadline misses.

### **TMan\_Ticks**

Esta função é o trabalho de execução da tarefa “ticks”. Para além de contar os TMan Ticks, esta tarefa tem a responsabilidade de ativar as restantes tarefas. Todos os TMan Ticks, percorre todas as tarefas adicionadas, verifica se é uma tarefa periódica ou uma tarefa esporádica. Esta verificação é feita através do atributo precedência.

Caso seja uma tarefa periódica, se o TMan Tick for maior ou igual que o nextActivation, esta tarefa pode ser ativada. Há ainda mais uma verificação a fazer que é verificar se já falhou a deadline, isto é verdade se o TMan Tick for maior que a currentActivation mais o período. Caso a deadline ainda não tenha sido violada, faz-se a ativação da tarefa (vTaskResume) e atualiza-se os atributos currentActivation, nextActivation, o contador de ativações e o estado é alterado para *Running*. Por outro lado, caso a deadline já tenha sido violada, a tarefa não conseguiu executar e atualiza-se o currentActivation e nextActivation e o contador de deadline misses.

Caso a tarefa seja esporádica, tem de se verificar se a tarefa que tem precedência já executou. Se a tarefa que precede está num estado de bloqueio e o TMan Tick em que acabou de executar é maior do que o TMan Tick da tarefa que tem de respeitar a precedência, então pode executar.

### **Task\_Work**

Esta função é o trabalho de execução das tarefas. É chamada a função TMan\_TaskWaitPeriod (explicada anteriormente) para a tarefa esperar pela próxima ativação. Assim que se der a ativação, é feita uma mensagem com o nome da tarefa que está a executar e com o tick da placa (xTaskGetTickCount). Seguidamente, foi feito dois loops *for* em cadeia, em que se realiza uma operação de soma com o objetivo de consumir algum tempo. Finalmente, no fim da execução, a tarefa é novamente posta num estado de bloqueio e guarda-se o TMan Tick em que acabou de executar.

### **TMan\_Print**

Esta função é o trabalho da tarefa responsável pelos “prints”. Acede à fila (xQueueReceive) onde as mensagens estão à espera e imprime a seguinte.

### **TMan\_Close**

Este método é responsável por terminar o *scheduler* (vTaskEndScheduler) e terminar tanto as tarefas criadas pelo utilizador como as tarefas de ticks e prints (vTaskDelete).

### **Testes e resultados**

De forma a testar as funcionalidades implementadas, foram sendo criadas várias tarefas com atributos que permitissem testar isso mesmo.

Um exemplo foi as tarefas criadas para testar o período das tarefas. Foram criadas tarefas onde o período delas era o dobro da anterior, ou seja, uma tarefa A com período 2, uma B com período 4 e uma C com período 8. Assim, sabe-se facilmente em que ticks cada tarefa tem de executar.

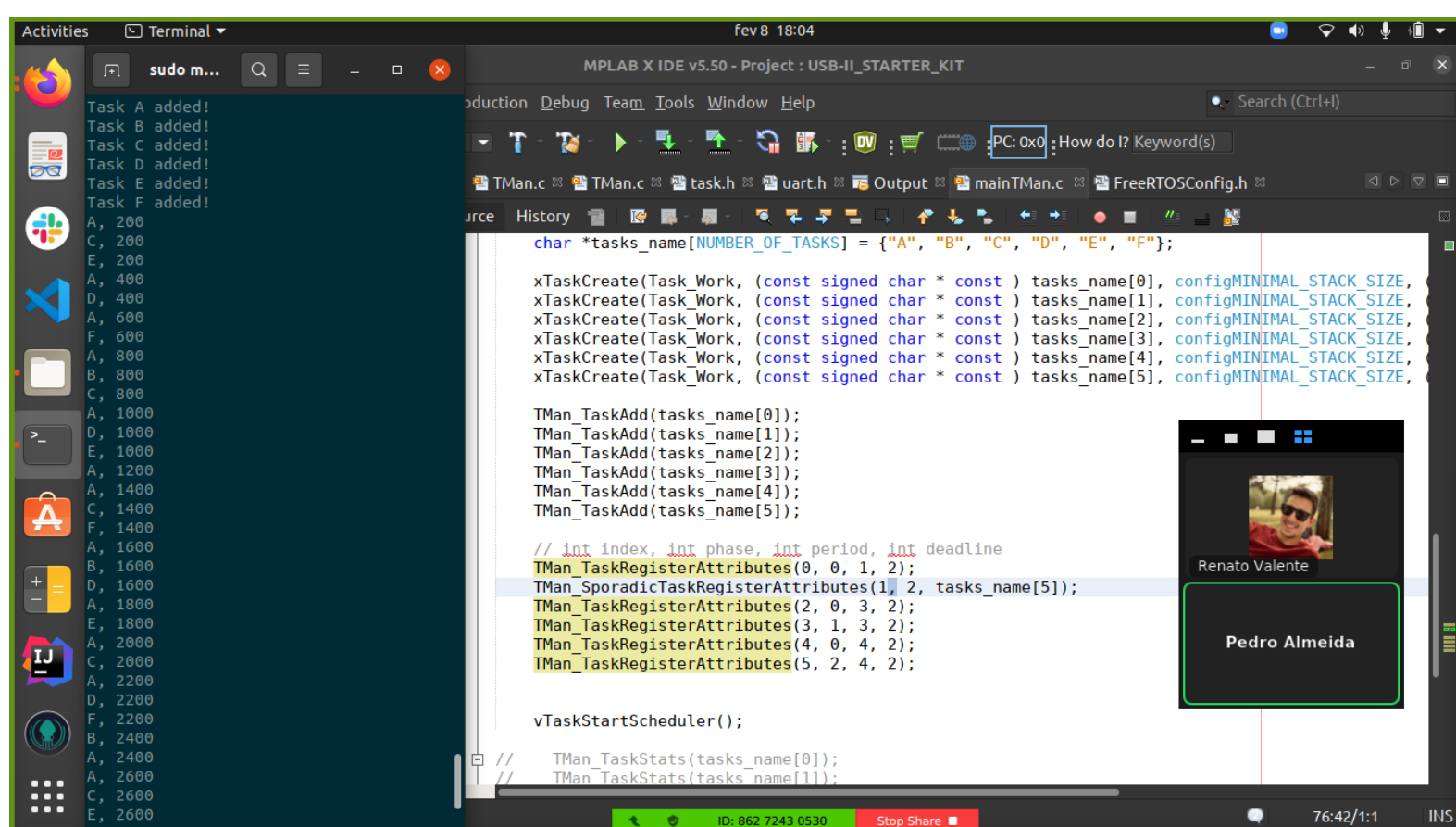
Para testar a fase, apenas é preciso uma tarefa e escolher um tick para começar.

Para testar a precedência foi criada duas tarefas onde a que tinha precedência tinha um período duas vezes mais rápido que a outra, e uma fase menor. Foi confirmado que a tarefa com precedência espera pela execução da tarefa que a precede.

### Alterações posteriores à demonstração

No dia da apresentação do trabalho, verificou-se que pelo menos um dos resultados obtidos não estavam 100% corretos. O que estava a acontecer era a tarefa *ticks* ao ter a maior prioridade, “roubava” o primeiro tick e as tarefas que tinham fase zero e deviam ser ativadas no primeiro instante, não estava a acontecer. Para resolver este problema foi adicionado um *delay* (*vTaskDelay*) com um tempo igual ao período.

Com a adição desta alteração obtivemos os seguintes resultados para os valores do teste 3:



```
Task A added!
Task B added!
Task C added!
Task D added!
Task E added!
Task F added!
A, 200
C, 200
E, 200
A, 400
D, 400
A, 600
F, 600
A, 800
B, 800
C, 800
A, 1000
D, 1000
E, 1000
A, 1200
A, 1400
C, 1400
F, 1400
A, 1600
B, 1600
D, 1600
A, 1800
E, 1800
A, 2000
C, 2000
A, 2200
D, 2200
F, 2200
B, 2400
A, 2400
A, 2600
C, 2600
E, 2600
```

```
char *tasks_name[NUMBER_OF_TASKS] = {"A", "B", "C", "D", "E", "F"};

xTaskCreate(Task_Work, (const signed char * const) tasks_name[0], configMINIMAL_STACK_SIZE,
xTaskCreate(Task_Work, (const signed char * const) tasks_name[1], configMINIMAL_STACK_SIZE,
xTaskCreate(Task_Work, (const signed char * const) tasks_name[2], configMINIMAL_STACK_SIZE,
xTaskCreate(Task_Work, (const signed char * const) tasks_name[3], configMINIMAL_STACK_SIZE,
xTaskCreate(Task_Work, (const signed char * const) tasks_name[4], configMINIMAL_STACK_SIZE,
xTaskCreate(Task_Work, (const signed char * const) tasks_name[5], configMINIMAL_STACK_SIZE,

TMan_TaskAdd(tasks_name[0]);
TMan_TaskAdd(tasks_name[1]);
TMan_TaskAdd(tasks_name[2]);
TMan_TaskAdd(tasks_name[3]);
TMan_TaskAdd(tasks_name[4]);
TMan_TaskAdd(tasks_name[5]);

// int index, int phase, int period, int deadline
TMan_TaskRegisterAttributes(0, 0, 1, 2);
TMan_SporadicTaskRegisterAttributes(1, 2, tasks_name[5]);
TMan_TaskRegisterAttributes(2, 0, 3, 2);
TMan_TaskRegisterAttributes(3, 1, 3, 2);
TMan_TaskRegisterAttributes(4, 0, 4, 2);
TMan_TaskRegisterAttributes(5, 2, 4, 2);

vTaskStartScheduler();

// TMan_TaskStats(tasks_name[0]);
// TMan_TaskStats(tasks_name[1]);
```

Durante a resolução do projeto, algumas questões relativamente à precedência, foram discutidas com o grupo do Pedro Valente (88858)