QXD0145 - Sistemas de Tempo-Real

Gerenciamento de *Tasks* no FreeRTOS I

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

André Ribeiro Braga

03/04/2023

Funções *task*

- Implementadas em linguagem C
- Protótipo padronizado
  - Retorna *void*
  - Ponteiro *void* como parâmetro
- Como um pequeno programa
- Dado um ponto de entrada
  - Normalmente tem laço infinito
  - Não retorna valor (sem *return*)
  - Caso não necessária, deve ser explicitamente deletada
- Uma mesma definição pode ser utilizada para várias *tasks*
  - Instâncias de execução separadas
  - Pilhas separadas

Funções *task*

Exemplo

```
void ATaskFunction( void *pvParameters )
{
/* Variables can be declared just as per a normal function.  Each instance of a task
created using this example function will have its own copy of the lVariableExample
variable.  This would not be true if the variable was declared static – in which case
only one copy of the variable would exist, and this copy would be shared by each
created instance of the task. (The prefixes added to variable names are described in
section 1.5, Data Types and Coding Style Guide.) */
int32_t lVariableExample = 0;

    /* A task will normally be implemented as an infinite loop. */
    for( ;; )
    {
        /* The code to implement the task functionality will go here. */
    }

    /* Should the task implementation ever break out of the above loop, then the task
    must be deleted before reaching the end of its implementing function.  The NULL
    parameter passed to the vTaskDelete() API function indicates that the task to be
    deleted is the calling (this) task.  The convention used to name API functions is
    described in section 0, Projects that use a FreeRTOS version older than V9.0.0
must build one of the heap_n.c files.  From FreeRTOS V9.0.0 a heap_n.c file is only
required if configSUPPORT_DYNAMIC_ALLOCATION is set to 1 in FreeRTOSConfig.h or if
configSUPPORT_DYNAMIC_ALLOCATION is left undefined.  Refer to Chapter 2, Heap Memory
Management, for more information.
Data Types and Coding Style Guide. */
    vTaskDelete( NULL );
}
```

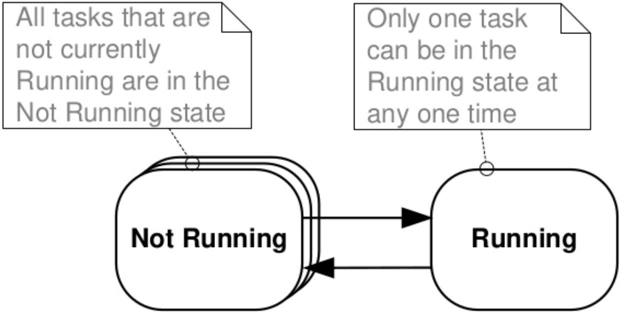
Estados de uma *task*

Alto Nível

- Uma aplicação pode consistir de vários estados
- Processadores com um núcleo $\Rightarrow$ Apenas uma *task* executa por vez
  - Estados *Running*/*Not Running*
- *Task* em *Running* $\Rightarrow$ Processador executando seu código
- *Task* em *Not Running* $\Rightarrow$ Dormente
  - Continua execução de acordo com escalonador
  - Instrução prestes a ser executada antes de sair do estado *Running*
- Transição *Not Running* $\rightarrow$ *Running*: *switched in*
- Transição *Running* $\rightarrow$ *Not Running*: *switched out*

Estados de uma *task*

Alto Nível



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Criando *tasks*

- Criadas utilizando a função *xTaskCreate* (mais complexa do API)

```
BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,
                        const char * const pcName,
                        uint16_t usStackDepth,
                        void *pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t *pxCreatedTask );
```

- Parâmetros
  - *pvTaskCode*: ponteiro para função
  - *pcName*: nome para fins de depuração
  - *usStackDepth*: tamanho da pilha
  - *pvParameters*: parâmetros da *task*
  - *uxPriority*: prioridade da *task*
  - *pxCreatedTask*: *handle* da *task*

- Retorno:
  - *pdPASS*: Sucesso
  - *pdFAIL*: Falha

Criando *tasks*

Exemplo 1

```c
int main( void )
{
    /* Create one of the two tasks.  Note that a real application should check
    the return value of the xTaskCreate() call to ensure the task was created
    successfully. */
    xTaskCreate(    vTask1,  /* Pointer to the function that implements the task. */
                    "Task 1",/* Text name for the task.  This is to facilitate
                             debugging only. */
                    1000,    /* Stack depth - small microcontrollers will use much
                             less stack than this. */
                    NULL,    /* This example does not use the task parameter. */
                    1,       /* This task will run at priority 1. */
                    NULL );  /* This example does not use the task handle. */

    /* Create the other task in exactly the same way and at the same priority. */
    xTaskCreate( vTask2, "Task 2", 1000, NULL, 1, NULL );

    /* Start the scheduler so the tasks start executing. */
    vTaskStartScheduler();

    /* If all is well then main() will never reach here as the scheduler will
    now be running the tasks.  If main() does reach here then it is likely that
    there was insufficient heap memory available for the idle task to be created.
    Chapter 2 provides more information on heap memory management. */
    for( ;; );
}
```

```c
void vTask1( void *pvParameters )
{
const char *pcTaskName = "Task 1 is running\r\n";
volatile uint32_t ul; /* volatile to ensure ul is not optimized away. */

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /* This loop is just a very crude delay implementation.  There is
            nothing to do in here.  Later examples will replace this crude
            loop with a proper delay/sleep function. */
        }
    }
}
```

```c
void vTask2( void *pvParameters )
{
const char *pcTaskName = "Task 2 is running\r\n";
volatile uint32_t ul; /* volatile to ensure ul is not optimized away. */

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /* This loop is just a very crude delay implementation.  There is
            nothing to do in here.  Later examples will replace this crude
            loop with a proper delay/sleep function. */
        }
    }
}
```

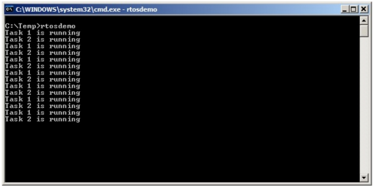
Criando *tasks*

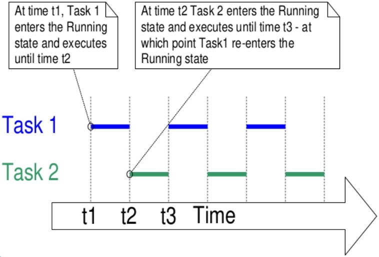
Exemplo 1

```c
void vTask1( void *pvParameters )
{
const char *pcTaskName = "Task 1 is running\r\n";
volatile uint32_t ul; /* volatile to ensure ul is not optimized away. */

    /* If this task code is executing then the scheduler must already have
    been started.  Create the other task before entering the infinite loop. */
    xTaskCreate( vTask2, "Task 2", 1000, NULL, 1, NULL );

    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /* This loop is just a very crude delay implementation.  There is
            nothing to do in here.  Later examples will replace this crude
            loop with a proper delay/sleep function. */
        }
    }
}
```

Criando *tasks*

Exemplo 1





Criando *tasks*

Exemplo 2

```
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;
volatile uint32_t ul; /* volatile to ensure ul is not optimized away. */

    /* The string to print out is passed in via the parameter.  Cast this to a
    character pointer. */
    pcTaskName = ( char * ) pvParameters;

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /* This loop is just a very crude delay implementation.  There is
            nothing to do in here.  Later exercises will replace this crude
            loop with a proper delay/sleep function. */
        }
    }
}
```

Criando *tasks*

Exemplo 2

```c
/* Define the strings that will be passed in as the task parameters.  These are
defined const and not on the stack to ensure they remain valid when the tasks are
executing. */
static const char *pcTextForTask1 = "Task 1 is running\r\n";
static const char *pcTextForTask2 = "Task 2 is running\r\n";

int main( void )
{
    /* Create one of the two tasks. */
    xTaskCreate(    vTaskFunction,          /* Pointer to the function that
                                            implements the task. */
                    "Task 1",               /* Text name for the task.  This is to
                                            facilitate debugging only. */
                    1000,                   /* Stack depth - small microcontrollers
                                            will use much less stack than this. */
                    (void*)pcTextForTask1,  /* Pass the text to be printed into the
                                            task using the task parameter. */
                    1,                      /* This task will run at priority 1. */
                    NULL );                 /* The task handle is not used in this
                                            example. */

    /* Create the other task in exactly the same way.  Note this time that multiple
    tasks are being created from the SAME task implementation (vTaskFunction).  Only
    the value passed in the parameter is different.  Two instances of the same
    task are being created. */
    xTaskCreate( vTaskFunction, "Task 2", 1000, (void*)pcTextForTask2, 1, NULL );

    /* Start the scheduler so the tasks start executing. */
    vTaskStartScheduler();

    /* If all is well then main() will never reach here as the scheduler will
    now be running the tasks.  If main() does reach here then it is likely that
    there was insufficient heap memory available for the idle task to be created.
    Chapter 2 provides more information on heap memory management. */
    for( ;; );
}
```

Prioridades de *Tasks*

- Parâmetro *uxPriority* atribui prioridades iniciais
- Pode ser modificada após o início do escalonador
  - Função *vTaskPrioritySet* do API
- Número máximo pode ser definido pela aplicação
  - *configMAX_PRIORITIES*
- Prioridade diretamente proporcional ao valor numérico
- *Tasks* distintas podem ter prioridades iguais

Prioridades de *Tasks*

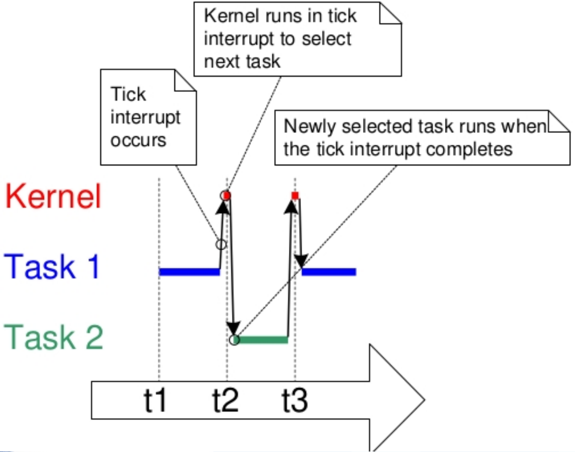
Métodos de decisão sobre execução

- Genérico
  - Implementado em C
  - Disponível para todos os *ports*
  - Não há limite para *configMAX_PRIORITIES*
  - Recomendável valor mínimo (maior valor $\Rightarrow$ mais RAM e WCET)
  - *configUSE_PORT_OPTIMISED_TASK_SELECTION*: zero ou indefinido
- Otimizado para a arquitetura
  - Código assembler
  - Mais rápido que o genérico
  - *configMAX_PRIORITIES* não afeta WCET (≤ 32)
  - Também recomendável valor mínimo (consumo de RAM)
  - *configUSE_PORT_OPTIMISED_TASK_SELECTION*: 1

Medidas Temporais e Interrupção de *Tick*

- *time slice*
  - Empregado nos exemplos anteriores
  - Todas *tasks* sempre prontas ⇒ Executadas por uma fatia temporal
- Escalonador sempre executa ao final do *time slice*
- Iterrupção periódica chamada *tick interrupt* utilizada para tal
- Tamanho do *time slice* definido pela frequência de *tick*
  - *configTICK_RATE_HZ*
  - Exemplo: *configTICK_RATE_HZ* 100 (Hz) ⇒ *time slice* 10ms
  - Tempo entre dois *ticks* igual ao período de *tick*

Medidas Temporais e Interrupção de *Tick*



Medidas Temporais e Interrupção de *Tick*

Exemplo

```c
/* Define the strings that will be passed in as the task parameters.  These are
defined const and not on the stack to ensure they remain valid when the tasks are
executing. */
static const char *pcTextForTask1 = "Task 1 is running\r\n";
static const char *pcTextForTask2 = "Task 2 is running\r\n";

int main( void )
{
    /* Create the first task at priority 1.  The priority is the second to last
    parameter. */
    xTaskCreate( vTaskFunction, "Task 1", 1000, (void*)pcTextForTask1, 1, NULL );

    /* Create the second task at priority 2, which is higher than a priority of 1.
    The priority is the second to last parameter. */
    xTaskCreate( vTaskFunction, "Task 2", 1000, (void*)pcTextForTask2, 2, NULL );

    /* Start the scheduler so the tasks start executing. */
    vTaskStartScheduler();

    /* Will not reach here. */
    return 0;
}
```

Medidas Temporais e Interrupção de *Tick*

Exemplo