Engenharia de Computação

Sistemas Operacionais Embarcados

Sistema Operacional FreeRTOS

Prof. Anderson Luiz Fernandes Perez
Universidade Federal de Santa Catarina
Campus Araranguá

Email: anderson.perez@ufsc.br

Conteúdo

- Introdução
- Gerenciamento de Tarefas
- Gerenciamento de Memória
- Variações (licenças)

Introdução

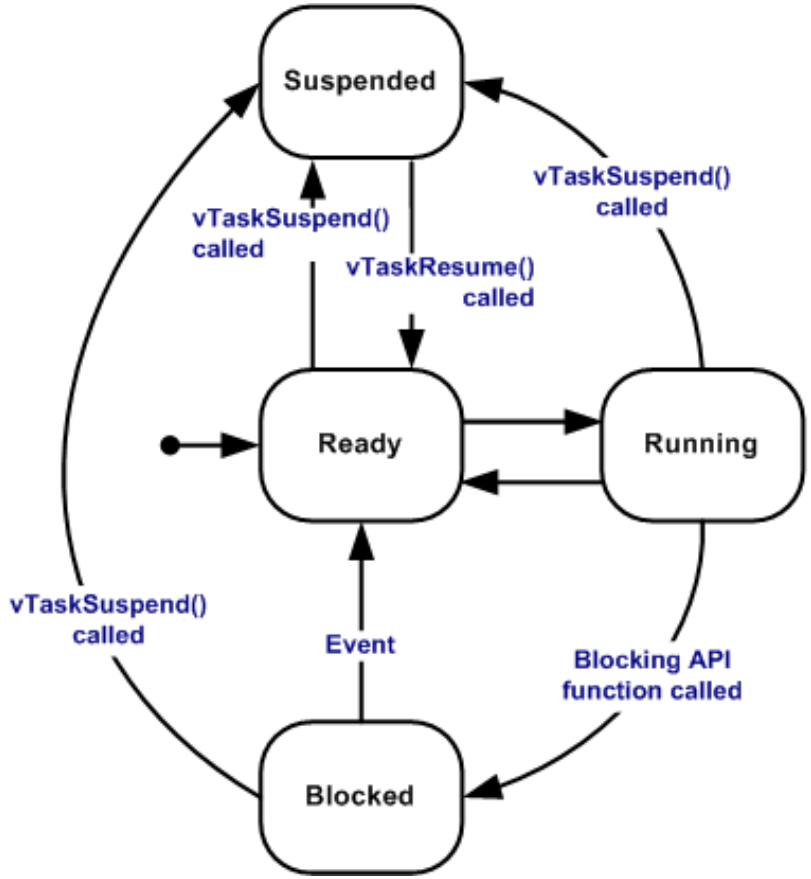
- O FreeRTOS é um sistema operacional de código aberto desenvolvido pela empresa *Real Time Engineers*.
- Foi projetado para ser executado em pequenos dispositivos computacionais.
- O FreeRTOS possui versões (ports) para mais de 23 arquiteturas de microcontroladores.
- O sistema foi desenvolvido em C e conta com uma API (*Application Programming Interface*) que disponibiliza funções para o gerenciamento de tarefas e o gerenciamento de memória.

Gerência de Tarefas

- O FreeRTOS permite a criação ilimitada de tarefas. O que limita a quantidade de tarefas é o hardware, capacidade de processamento e memória.
- Na API do sistema existem várias funções para manipulação de tarefas, tais como:
  - *vTaskCreate() – para a criação de tarefas*
  - *vTaskDelete() – para a destruição de tarefas*
  - *vTaskSuspend() – para suspender a execução de uma tarefa*
  - *vTaskResume() – para retomar a execução de uma tarefa suspensa.*

Gerência de Tarefas

- Ciclo de vida de uma Tarefa



Gerência de Tarefas

- Definição de Tarefas
  - Uma tarefa no FreeRTOS é uma função na linguagem de programação C que tem uma parâmetro do tipo *void** e como tipo de retorno *void*.
  - Exemplo de uma tarefa:

```
void piscaLedTask(void* parameters);
```

Gerência de Tarefas

- Criação de uma Tarefa
  - A função para criação de uma tarefa é:
    - Biblioteca *task.h*

BaseType_t **xTaskCreate**(*TaskFunction_t pvTaskCode, const char * const pcName, unsigned short usStackDepth, void *pvParameters, UBaseType_t uxPriority, TaskHandle_t *pvCreatedTask*);

Gerência de Tarefas

- Criação de uma Tarefa
  - Parâmetros de *xTaskCreate()* (1/2)
    - pvTaskCode: ponteiro para a função que implementa a tarefa.
    - pcName: um nome qualquer dado para a tarefa. Geralmente usado na depuração.
    - usStackDepth: tamanho da pilha da tarefa. O parâmetro é passado em words.

Gerência de Tarefas

- Criação de uma Tarefa
  - Parâmetros de *xTaskCreate()* (2/2)
    - pvParameters: um ponteiro para os argumentos da tarefa. Se for necessário passar mais de uma parâmetro para a tarefa estes devem estar encapsulados em uma estrutura.
    - uxPriority: prioridade da tarefa. A prioridade deve um número inteiro entre 0 e MAX_PRIORITIES – 1.
    - pvCreatedTask: um ponteiro para um manipulador da tarefa. Caso não seja utilizado, deve ser deixado como NULL. Este manipulador é utilizado em outras rotinas do gerenciador de tarefas, tal como a função *vTaskDelete()*.

Gerência de Tarefas

- Destruição de uma Tarefa
  - A função para destruir uma tarefa é:
    - Biblioteca *task.h*

```
void vTaskDelete( TaskHandle_t xTask );
```

- O parâmetro *xTask* é o manipulador da tarefa.
- Se a tarefa tiver realizado alguma alocação dinâmica de memória, ela deve liberar o espaço reservado antes da destruição.

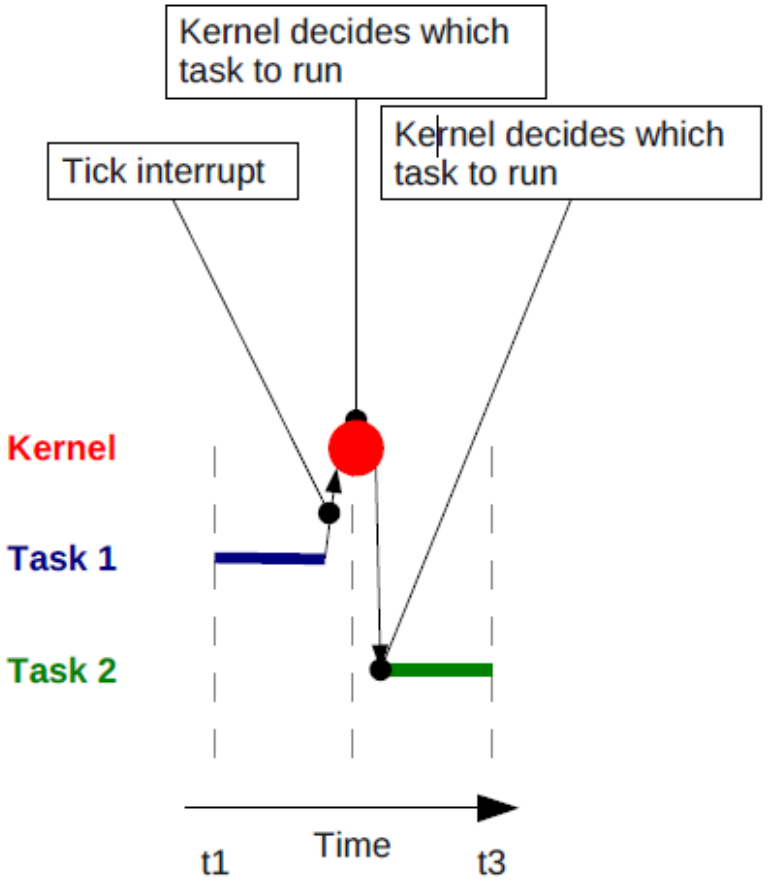
Gerência de Tarefas

- Escalonamento de Tarefas
  - As tarefas no FreeRTOS são escalonadas de acordo com a prioridade definida em cada tarefa.
  - As prioridades são definidas de 0 até *MAX_PRIORITIES*, constante definida no arquivo *FreeRTOSConfig.h*.
  - A prioridade 0 (zero) é reservada para a tarefa *idle*.
  - Número menores representam maior prioridade e vice-versa, ou seja, as prioridades são inversamente proporcionais as números que as definem.

Gerência de Tarefas

- Escalonamento de Tarefas

A cada *tick* do relógio o escalonador de tarefas decide qual a próxima tarefa deverá entrar em execução.

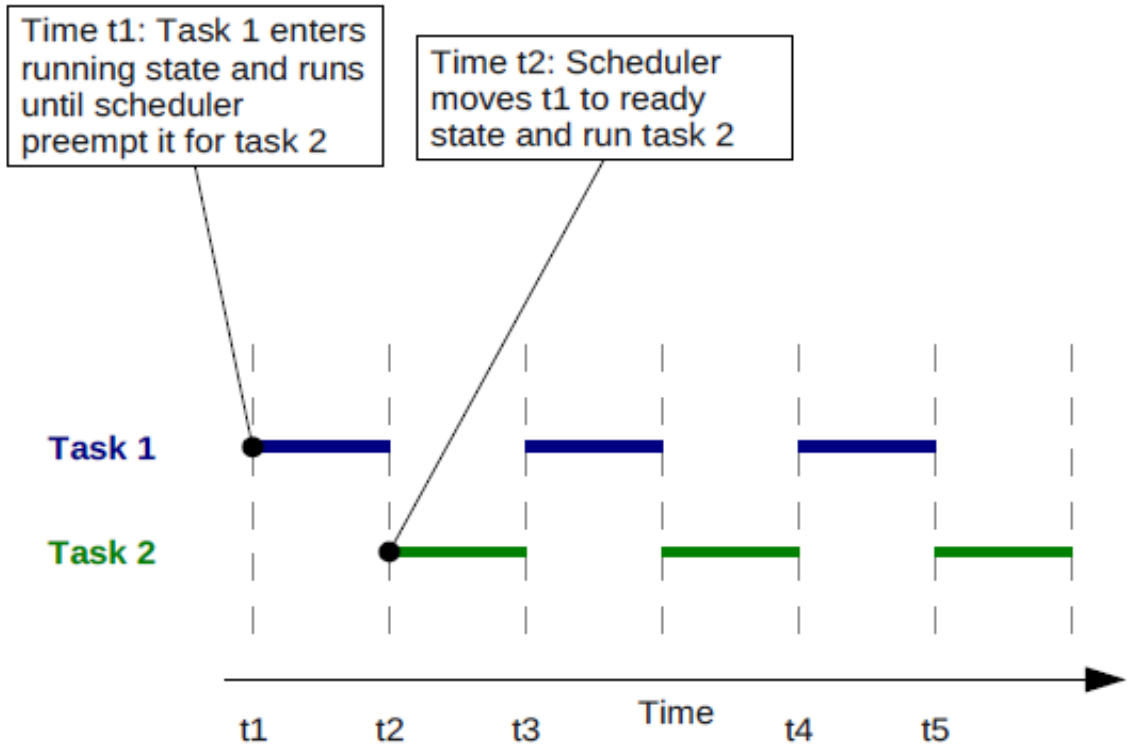


Gerência de Tarefas

- Escalonamento de Tarefas
  - Tarefas que possuem a mesma prioridade são escalonadas com o algoritmo *Round-Robin*.
  - O quantum do algoritmo RR é definido conforme o tempo de cada *tick* de relógio.
  - O tempo do *tick* de relógio é definido na constante TICK_RATE_HZ no arquivo *FreeRTOSconfig.h*.

Gerência de Tarefas

- Escalonamento de Tarefas
  - Exemplo de Escalonamento com RR



Gerência de Tarefas

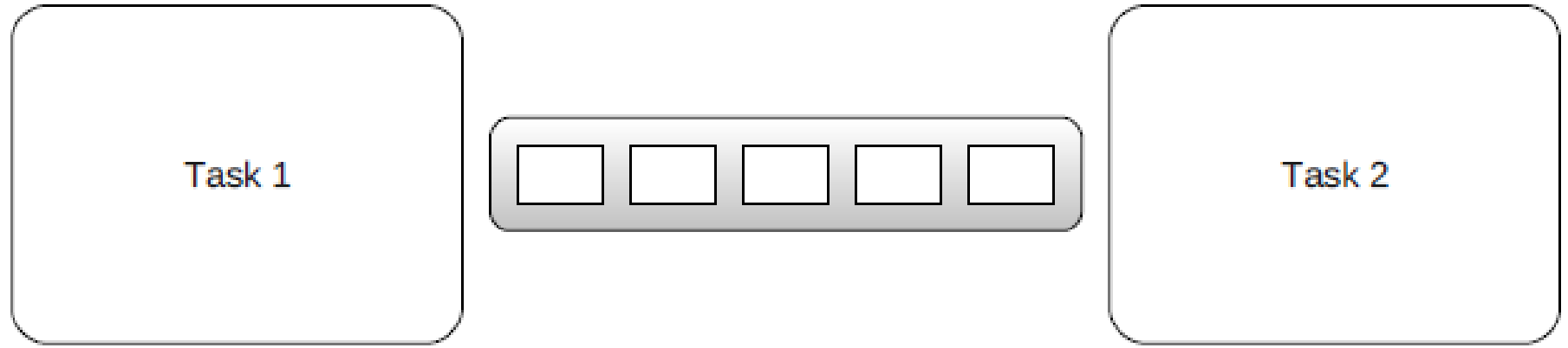
- Starvation/Inanição
  - O FreeRTOS **não possui** nenhum mecanismo para evitar *starvation*.
  - O programador deve ficar atento quando definir tarefas que executam todo o tempo e que possuem ata prioridade no sistema.
  - Uma boa prática de programação é permitir que a tarefa *idle* entre em execução algumas vezes.

Gerência de Tarefas

- Fila de Mensagens
  - Uma fila é um mecanismo fornecido pelo sistema operacional para que duas tarefas possam se comunicar.
  - Um fila é implementada como uma FIFO, ou seja, os dados lidos seguem a mesma sequência da escrita.

Gerência de Tarefas

- Fila de Mensagens
  - Exemplo: fila para cinco mensagens



Gerência de Tarefas

- Fila de Mensagens
  - Criação de uma Fila de Mensagens
    - Biblioteca: *queue.h*

```
xQueueHandle xQueueCreate( unsigned portBASE_TYPE uxQueueLength,
unsigned portBASE_TYPE uxItemSize
);
```

- Onde:
  - **uxQueueLength** define o tamanho da fila.
  - **uxItemSize** define o tamanho de cada item da fila.

Gerência de Tarefas

- Fila de Mensagens
  - Ler dados de em uma Fila de Mensagens
    - Biblioteca: *queue.h*

```
portBASE_TYPE xQueueReceive( xQueueHandle xQueue, const void * pvBuffer,
portTickType xTicksToWait
);
```

- Onde:
  - **uxQueue** é o descritor da fila.
  - **pvBuffer** é o endereço do buffer para onde o dados será copiado.
  - **xTickToWait** define o tempo máximo que a tarefa ficará bloqueada na fila. O valor 0 (zero) indicará que a tarefa ficará bloqueada indefinidamente.

Gerência de Tarefas

- Fila de Mensagens
  - Escrever dados em uma Fila de Mensagens
    - Biblioteca: *queue.h*

```
portBASE_TYPE xQueueSend( xQueueHandle xQueue, const void *
pvItemToQueue, portTickType xTicksToWait
);
```

- Onde:
  - **uxQueue** é o descritor da fila.
  - **pvItemToQueue** endereço de um elemento que está aguardando ser copiado para a fila.
  - **xTicksToWait** tempo que a tarefa ficará aguardando após ter copiado o dado para a fila.

Gerência de Tarefas

- Sincronização de Tarefas
  - O FreeRTOS permite a sincronização de tarefas a partir do uso de:
    - Semáforos binários
    - Semáforos contadores
    - Variáveis mutex.

Gerência de Tarefas

- Sincronização de Tarefas
  - Semáforos Binários
    - Criação de semáforos binários
      - Biblioteca *semphr. h*

```
void vSemaphoreCreateBinary( xSemaphoreHandle xSemaphore );
```

      - Onde:
        - » **xSemaphore** é o descritor do semáforo.

Gerência de Tarefas

- Sincronização de Tarefas
  - Semáforos Binários
    - Obtendo/Trabando um semáforo binário
      - Biblioteca *semphr. h*

portBASE_TYPE xSemaphoreTake(*xSemaphoreHandle xSemaphore, portTickType xTicksToWait*);

      - Onde:
        - » **xSemaphore** é o descritor do semáforo.
        - » **xTicksToWait** é o tempo em que a tarefa ficará bloqueada no semáforo.

Gerência de Tarefas

- Sincronização de Tarefas
  - Semáforos Binários
    - Liberando um semáforo binário
      - Biblioteca *semphr. h*

```
portBASE_TYPE xSemaphoreGive( xSemaphoreHandle xSemaphore );
```

      - Onde:
        - » **xSemaphore** é o descritor do semáforo.

Gerência de Tarefas

- Sincronização de Tarefas
  - Semáforos Contadores
    - Criação de semáforos contador
      - Biblioteca *semphr. h*

xSemaphoreHandle xSemaphoreCreateCounting(*unsigned portBASE_TYPE uxMaxCount, unsigned portBASE_TYPE uxInitialCount*);

      - Onde:
        - » **uxMaxCount** valor do contador do semáforo.
        - » **uxInitialCount** valor inicial do semáforo.

Gerência de Tarefas

- Sincronização de Tarefas
  - Semáforos Contadores
    - Obtendo/Trabando um semáforo contador
      - Biblioteca *semphr. h*

portBASE_TYPE xSemaphoreTake(*xSemaphoreHandle xSemaphore, portTickType xTicksToWait*);

      - Onde:
        - » **xSemaphore** é o descritor do semáforo.
        - » **xTicksToWait** é o tempo em que a tarefa ficará bloqueada no semáforo.

Gerência de Tarefas

- Sincronização de Tarefas
  - Semáforos Contadores
    - Liberando um semáforo contador
      - Biblioteca *semphr. h*

```
portBASE_TYPE xSemaphoreGive( xSemaphoreHandle xSemaphore );
```

      - Onde:
        - » **xSemaphore** é o descritor do semáforo.

Gerência de Tarefas

- Sincronização de Tarefas
  - Mutex
    - Mutex e semáforos binários são semelhantes na forma de bloquear a seção crítica.
    - No FreeRTOS um mutex implementa a inversão de prioridade e um semáforo binário não.
    - Semáforos binários são recomendados para prover sincronização entre tarefas.
    - Um mutex é indicado para prover um mecanismo simples de exclusão mútua.

Gerência de Tarefas

- Sincronização de Tarefas
  - Mutex
    - Criação de um mutex
      - Biblioteca *semphr. h*

```
SemaphoreHandle_t xSemaphoreCreateMutex( void )
```

Gerência de Tarefas

- Sincronização de Tarefas
  - Mutex
    - Obtendo/Trabando mutex
      - Biblioteca *semphr. h*

portBASE_TYPE xSemaphoreTake(*xSemaphoreHandle xSemaphore, portTickType xTicksToWait*);

      - Onde:
        - » **xSemaphore** é o descritor do semáforo.
        - » **xTicksToWait** é o tempo em que a tarefa ficará bloqueada no semáforo.

Gerência de Tarefas

- Sincronização de Tarefas
  - Mutex
    - Liberando um mutex
      - Biblioteca *semphr. h*

```
portBASE_TYPE xSemaphoreGive( xSemaphoreHandle xSemaphore );
```

      - Onde:
        - » **xSemaphore** é o descritor do semáforo.

Gerência de Tarefas

- Inversão de Prioridades
  - No FreeRTOS quando duas ou mais tarefas estiverem solicitando uma semáforo ou variável mutex o sistema dá preferência para as tarefas de mais alta prioridade, evitando assim a inversão de prioridade.

Gerência de Tarefas

- Seção Crítica
  - O FreeRTOS possui duas maneiras de proteger a seção crítica, além dos semáforos e mutex.
  - A primeira consiste em desabilitar todas as interrupções e habilitá-las novamente ao final da operação crítica que se quer realizar.
  - A segunda consiste em desabilitar o escalonador de tarefas.

Gerência de Tarefas

- Seção Crítica
  - Controlando as Interrupções

```
taskENTER_CRITICAL()

// seção crítica

taskEXIT_CRITICAL()
```

Gerência de Tarefas

- Seção Crítica
  - Desabilitando o Escalonador de Tarefas

```
vTaskSuspendAll()

// seção crítica

vTaskResumeAll()
```

Gerência de Memória

- O FreeRTOS aloca memória quando os seguintes eventos acontecem (principais):
  - Criação de filas;
  - Criação de semáforos e variáveis mutex;
  - Quando se estabelece um temporizador de software;
  - ...
- As função malloc() e free() implementadas no padrão ANSI C não podem ser usadas, devido a:
  - As vezes não estão disponíveis para alguns sistemas embarcados;
  - A biblioteca ocupa espaço de memória;
  - Não serem thread-safe;
  - Não serem determinísticas.

Gerência de Memória

- O gerenciamento de memória no FreeRTOS pode ser implementado de 5 (cinco) maneiras diferentes, a depender o hardware que se está utilizando.
- As cinco formas de gerenciar e memória são:
  - **heap_1** – implementação simples, não permite liberação de memória.
  - **heap_2** – permite liberação de memória, mas não permite a combinação de blocos livres adjacentes para formar um bloco maior.
  - **heap_3** – faz com que malloc() e free() sejam thread-safe.
  - **heap_4** – permite a combinação de blocos adjacentes libres para evitar a fragmentação da memória.
  - **heap_5** – como o heap_4, porém permite estender a heap em diversas área de memória não adjacentes.

Gerência de Memória

- Protótipo das Funções *malloc()* e *free()*

```
void *pvPortMalloc( size_t xWantedSize);
```

```
void pvPortFree( void *pv);
```

Gerência de Memória

- Exemplo das Funções *malloc()* e *free()* Implementadas como Thread-Safe

```
void *pvPortMalloc( size_t xWantedSize )
{
  void *pvReturn;
  vTaskSuspendAll();
  {
      pvReturn = malloc( xWantedSize );
  }
  xTaskResumeAll();
  return pvReturn;
}
```

```
void vPortFree( void *pv )
{
  if( pv != NULL ) {
    vTaskSuspendAll();
    {
      free( pv );
    }
    xTaskResumeAll();
  }
}
```

Variações (licenças)

- O FreeRTOS possui duas variações mantidas pela empresa *HighIntegritySystems*:
  - SafeRTOS - versão certificada para aplicações críticas, tais como: veicular, aeroespacial, médica etc.
  - OpenRTOS – versão comercial com documentação, suporte e treinamento.