Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento Acadêmico de Eletrônica (DAELN)

SISTEMAS EMBARCADOS

Controle de Kernel, gerenciamento de Threads e funções de espera

Prof. André Schneider de Oliveira

andreoliveira@utfpr.edu.br

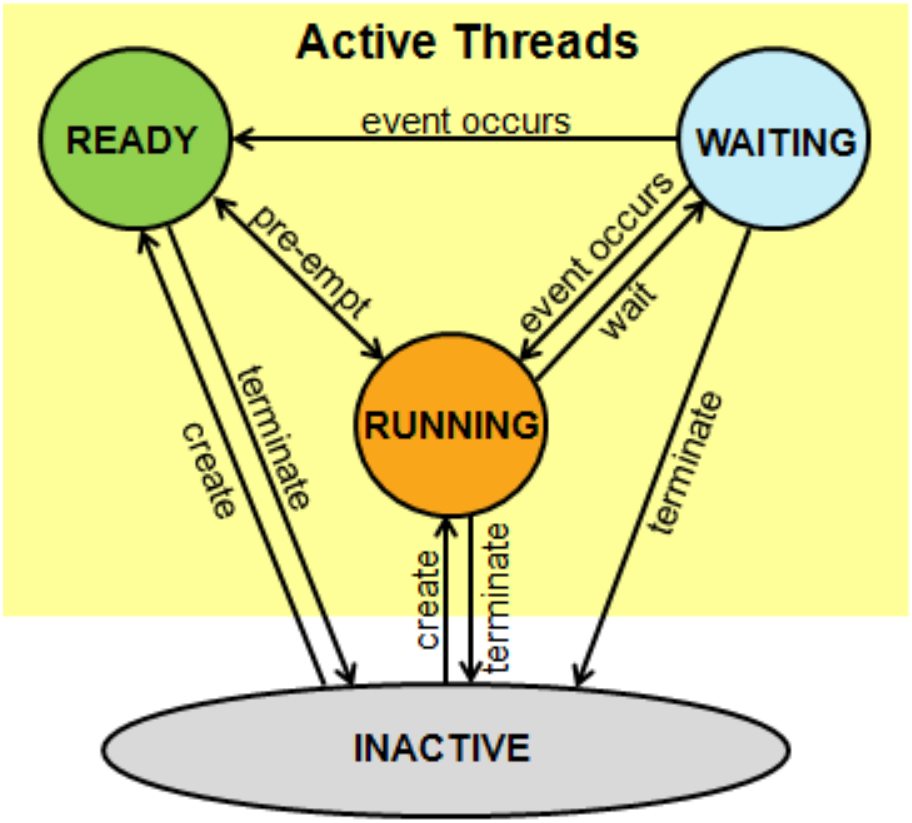
Kernel

- Kernel (ou núcleo) é uma abstração do hardware para a programação em alto-nível das interfaces

- Promove a conexão entre hardware e software por
  - processos
  - comunicação entre processos
  - memória virtual
  - sistema de arquivos

- Contém um conjunto de "**device drivers**" para gerenciar a interação com os subsistemas de hardware

Tarefas do Kernel



Thread State and State Transitions

Kernel

Através do Kernel RTOS é possível

- obter informações do sistema e do Kernel
- obter a versão do CMSIS-RTOS API
- inicializar o Kernel e criar os objetos RTOS
- iniciar a execução do Kernel RTOS e do gerenciamento das threads
- verificar o status da execução do Kernel RTOS

Kernel

A função **Main** é uma thread especial que é iniciada juntamente com o Kernel com prioridade inicial **osPriorityNormal**

Na função **Main** é preciso seguir a seguinte estrutura:

1. Chamar **osKernelInitialize()** para inicializar o Kernel CMSIS-RTOS
2. Configurar os periféricos, criar variáveis e definir os objetos RTOS
3. Chamar **osKernelStart()** para iniciar o Kernel e o gerenciamento de threads

```
int main (void) {
  osKernelInitialize ();                    // initialize CMSIS-RTOS

  // initialize peripherals here

  // create 'thread' functions that start executing,
  // example: tid_name = osThreadCreate (osThread(name), NULL);

  osKernelStart ();                         // start thread execution
}
```

Informação e Controle do Kernel

- **É possíver testar a correta inicialização do Kernel**
  - **osKernelInitialize() - inicializa o kernel**
  - **osKernelStart() -** Ativa o kernel
  - **osKernelRunning() -** verifica se o kernel está ativo
    - **0 - RTOS não ativo**
    - **1 - RTOS ativo**

```
#include "cmsis_os.h"

int main (void)  {
  if (osKernelInitialize () != osOK)  {          // check osStatus for other possible valid values
    // exit with an error message
  }

  if (!osKernelRunning ())  {                    // is the kernel running ?
    if (osKernelStart () != osOK)  {             // start the kernel
                                                 // kernel could not be started
    }
  }
}
```

Informação e Controle do Kernel

- O CMSIS RTOS emprega o **System Tick Timer (SysTick)** para gerar pedidos de interrrupção regulares, possibilitando a preempção das tarefas

- O SysTick pode ser utilizado para gerar atrasos, medir o tempo ou como uma fonte de interrupção para tarefas periódicas

osKernelSysTick() - Obtém o valor do temporizador do kernel

– retorna o tempo em um valor 32-bits

```
#include "cmsis_os.h"
void SetupDevice (void)  {
  uint32_t tick;

  tick = osKernelSysTick();                    // get start value of the Kernel system tick
  Device.Setup ();                             // initialize a device or peripheral
  do {                                         // poll device busy status for 100 microseconds
    if (!Device.Busy) break;                   // check if device is correctly initialized
  } while ((osKernelSysTick() - tick) < osKernelSysTickMicroSec(100));
  if (Device.Busy)  {
    ;                                          // in case device still busy, signal error
  }
                                               // start interacting with device
}
```

Controle de Kernel no RTOS

Macros (ou definições possíveis) - "cmsis_os.h"

#define **osFeature_MainThread** 1

- define ser a função **Main** será uma thread **1=habilitada, 0=desabilitada**

#define **osFeature_SysTick** 1

- habilita as funções do osKernelSysTick **1=habilitada, 0=desabilitada**

#define **osCMSIS** 0x10002 - versão da API CMSIS

#define **osCMSIS_KERNEL** 0x10000 - Identificação e versão do RTOS

#define **osKernelSystemId** "KERNEL V1.00" - String de identificação do RTOS

#define **osKernelSysTickFrequency** 100000000 - Frequência (Hz) do SysTick

#define **osKernelSysTickMicroSec(microsec)**
(((uint64_t)microsec * (osKernelSysTickFrequency)) / 1000000)

- Converte um tempo em microssegundos para a frequência do SysTick
- Comumente utilizado para pequenos atrasos em tarefas de **"pooling"**

Configurações do Kernel

Arquivo **RTX_Conf_CM.c**

Configurações das threads

- **OS_TASKCNT** = número máximo de threads executando concorrentemente <padrão 6>
- **OS_STKSIZE** = define o tamanho da pilha threads com stackz=0 <padrão 200>
- **OS_MAINSTKSIZE** = define o tamanho da pilha para a Main thread <padrão 200>
- **OS_PRIVCNT** = # threads com pilhas especificadas pelo usuário <padrão 0>
- **OS_STKCHECK** = habilita ou desabilita o teste de **"overflow"** de pilha na preempção (essa opção atrasa a troca de threads)
- **OS_RUNPRIV** = define o modo de execução das threads (0=Unpriv, 1=Priv) <padrão 1>

Configurações do Kernel

Arquivo **RTX_Conf_CM.c**

Configurações do SysTick

- **OS_SYSTICK** = 1 para utilizar o SysTick timer como RTOS Kernel Timer

- **OS_CLOCK** = especifica a frequência do RTOS Kernel timer [Hz], geralmente é idêntico ao **core clock**

- **OS_TICK** = intervalo do Systick [µs] <padrão 1000 = 1µs>

Função de idle

void os_idle_demon (void) {....}

função que é executada quando nenhuma thread está em estado **ready**

Gerenciamento de Threads

- **`osThreadCreate`** – Ativa a execução de uma tarefa
- **`osThreadTerminate`** – Desativa a execução de uma tarefa
- **`osThreadYield`** – Passa a execução à próxima tarefa que pronta
- **`osThreadGetId`** – Obtém o identificador que referencia a tarefa
- **`osThreadSetPriority`** – Altera a prioridade de uma tarefa
- **`osThreadGetPriority`** – Obtém a prioridade atual de uma tarefa

Gerenciamento de Threads

osStatus **osThreadTerminate** (osThreadId thread_id)

- Desativa a execução de uma tarefa
- Retorna o status da solicitação

Status and Error Codes

- osOK = thread finalizada com sucesso
- osErrorParameter = thread_id está incorreta
- osErrorResource = thread_id é de uma thread não ativa
- osErrorISR = não pode ser chamada de uma ISR

```
#include "cmsis_os.h"

void Thread_1 (void const *arg);                          // function prototype for Thread_1
osThreadDef (Thread_1, osPriorityNormal, 1, 0);           // define Thread_1
void ThreadTerminate_example (void) {
  osStatus status;
  osThreadId id;

  id = osThreadCreate (osThread (Thread_1), NULL);        // create the thread
  :
  status = osThreadTerminate (id);                        // stop the thread
  if (status == osOK) {
    // Thread was terminated successfully
  }
  else {
    // Failed to terminate a thread
  }
}
```

Gerenciamento de Threads

osStatus **osThreadYield** ()

- Passa o contole para a próxima thread pronta
- Caso não exista outra thread **ready**, continua executando a thread atual
- Retorna o status da solicitação

Status and Error Codes

- osOK = função executada corretamente
- osErrorISR = não pode ser chamada de uma ISR

```
#include "cmsis_os.h"

void Thread_1 (void const *arg)  {                          // Thread function
  osStatus   status;                                        // status of the executed function
  :
  while (1)  {
    status = osThreadYield();                               //
    if (status != osOK)  {
      // thread switch not occurred, not in a thread function
    }
  }
}
```

Gerenciamento de Threads

osThreadId **osThreadGetId** ()

- Busca o identificador uma thread
- Retorna o o thread_id ou NULL se for incorreto

```
void ThreadGetId_example (void)  {
  osThreadId id;                                    // id for the currently running thread

  id = osThreadGetId ();
  if (id == NULL) {
    // Failed to get the id; not in a thread
  }
}
```

Gerenciamento de Threads

osPriority **osThreadGetPriority** (osThreadId thread_id)

- Busca a prioridade de uma thread
- Retorna o a prioridade

```c
#include "cmsis_os.h"

void Thread_1 (void const *arg)  {                    // Thread function
  osThreadId id;                                      // id for the currently running thread
  osPriority priority;                                // thread priority

  id = osThreadGetId ();                              // Obtain ID of current running thread

  if (id != NULL)  {
    priority = osThreadGetPriority (id);
  }
  else  {
    // Failed to get the id
  }
}
```

Gerenciamento de Threads

osStatus **osThreadSetPriority** (osThreadId thread_id, osPriority priority)

- Muda a prioridade de uma thread
- O thread_id pode ser obtido do osThreadCreate or osThreadGetId
- Retorna o status da solicitação

Status and Error Codes

- osOK = prioridade alterada
- osErrorParameter = thread_id está incorreta
- osErrorValue = o valor de prioridade está incorreto
- osErrorResource = thread_id é de uma thread não ativa
- osErrorISR = não pode ser chamada de uma ISR

```
#include "cmsis_os.h"

void Thread_1 (void const *arg)  {                          // Thread function
  osThreadId id;                                            // id for the currently running thread
  osPriority pr;                                            // thread priority
  osStatus   status;                                        // status of the executed function

  :
  id = osThreadGetId ();                                    // Obtain ID of current running thread

  if (id != NULL) {
    status = osThreadSetPriority (id, osPriorityBelowNormal);
    if (status == osOK)  {
      // Thread priority changed to BelowNormal
    }
    else {
      // Failed to set the priority
    }
  }
  else  {
    // Failed to get the id
  }
  :
}
```

Funções de espera

1. **osDelay:** Suspende a execução de uma thread por um intervalo de tempo

2. **osWait:** Aguarda um evento não especificado por um período de tempo

Macros (ou definições possíveis) - "cmsis_os.h"

- #define **osFeature_Wait** 1
  - 1=habilitado, 0=não habilitado

Funções de espera

osStatus **osDelay** (uint32_t millisec)

- Retorna o status da solicitação

Status and Error Codes

- osEventTimeout = delay executado
- osErrorISR = não pode ser chamada de uma ISR

```
#include "cmsis_os.h"

void Thread_1 (void const *arg)  {                  // Thread function
  osStatus status;                                  // capture the return status
  uint32_t delayTime;                               // delay time in milliseconds

  delayTime = 1000;                                 // delay 1 second
  :
  status = osDelay (delayTime);                     // suspend thread execution
    // handle error code
  :
}
```

Funções de espera

osEvent **osWait** (uint32_t millisec)

- Aguarda um evento não especificado
- milisec: timeout ou 0 para sem timeout
- Retorna o evento com um componente RTOS ou erro

Eventos

- Um signal enviado para uma thread explicitamente
- Um mail ou message registrado para uma thread

Status and Error Codes

- osEventSignal = ocorreu um evento de signal
- osEventMessage = ocorreu um evento de message
- osEventMail = ocorreu um evento de mail
- osEventTimeout = não ocorreu um evento dentro do tempo especificado (timeout)
- osErrorISR = não pode ser chamada de uma ISR

Funções de espera

osEvent **osWait** (uint32_t millisec)

```
#include "cmsis_os.h"

void Thread_1 (void const *arg)  {                  // Thread function
  osEvent  Event;                                   // capture the event
  uint32_t waitTime;                                // wait time in milliseconds

  :
  waitTime = osWaitForever;                         // special "wait" value
  Event = osWait (waitTime);                        // wait forever and until an event occurred
  switch (Event.status)  {
    case osEventSignal:                             // Signal arrived
         :                                          // Event.value.signals contains the signal flags
      break;

    case osEventMessage:                            // Message arrived
         :                                          // Event.value.p contains the message pointer
         :                                          // Event.def.message_id contains the message Id
      break;

    case osEventMail:                               // Mail arrived
         :                                          // Event.value.p contains the mail pointer
         :                                          // Event.def.mail_id contains the mail Id
      break;

    case osEventTimeout:                            // Timeout occurred
      break;

    default:                                        // Error occurred
      break;
  }
  :
}
```