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

SISTEMAS EMBARCADOS

Sincronismo de threads e Temporizadores

Prof. André Schneider de Oliveira
andreoliveira@utfpr.edu.br

Comunicação entre threads

- Comunicação entre processos - Interprocess communication (IPC) consiste em um conjunto de mecanismos para troca de informações entre os processos

Existem dois tipos de comunicação

- Bloqueante = a informação é enviada e o processo fica aguardado resposta.
  - Ocorre a preempção para outro processo pronto (**ready**) e o processo atual vai para o estado de espera (**waiting**)

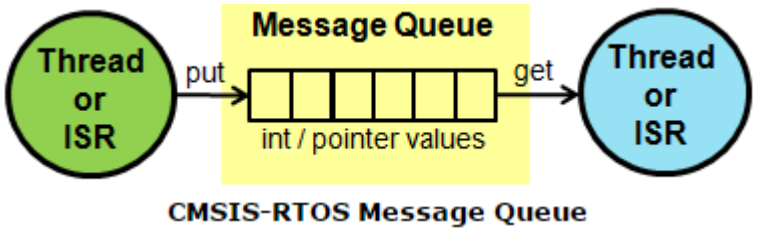
- Não bloqueante = a informação é enviada sem a necessidade de resposta
  - O processo continua em execução (**running**)

Comunicação entre threads

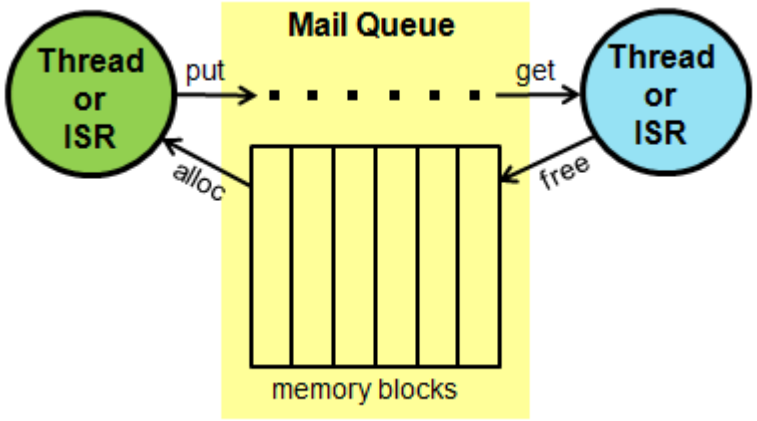
- Principais métodos para a comunicação entre processos
  - **eventos entre processos**
    - uso flags sinalizadoras de eventos entre os processos
    - um processo fica aguardando que seja realizado um evento de alteração de flag
  - **memória compartilhada**
    - os processos contém um espaço de memória em comum
    - devem haver politicas para evitar a perda/destruição das informações
  - **troca de mensagem**
    - os processos enviam mensagem em um canal de comunicação (real ou virtual)
    - não existe um endereçamento comum

Comunicação entre threads

- O CMSIS RTOS contém mecanismos para a comunicação entre threads
- Esse componentes são geradores de eventos (***osWait***)
  - sinais (flags sinalizadores)
  - mensagens (inteiro de 32 bits ou ponteiros)
  - correspondências (blocos de memória)



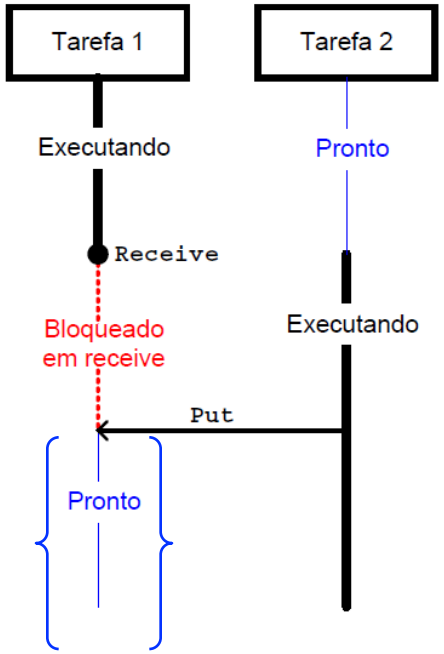
CMSIS-RTOS Message Queue



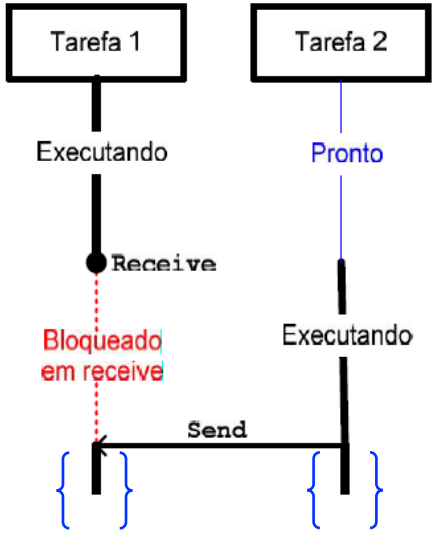
CMSIS-RTOS Mail Queue

Comunicação entre threads

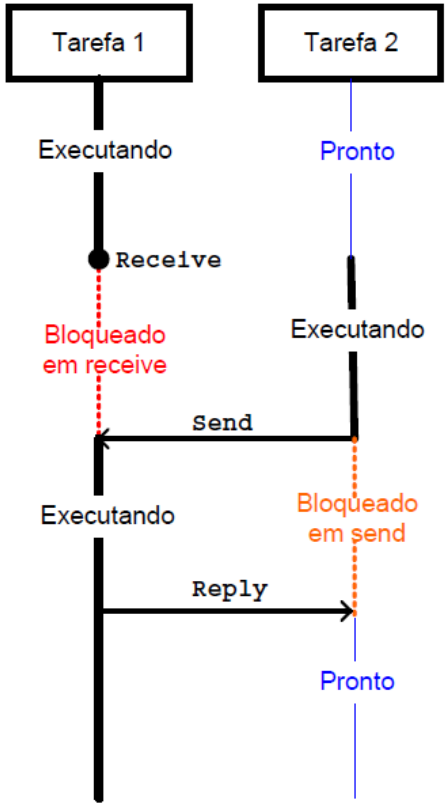
Assíncrono



Síncrono



Síncrono com resposta



Comunicação por eventos

- O signal (sinal) é uma flag compartilhada entre duas threads que gera eventos como uma *interrupção de software*
- O evento de signal é análogo à uma interrupção e força a execução de uma determinada área de um processo
- Essa modalidade de comunicação não tem o intercâmbio de dados mas de eventos (do tipo do signal)
- Os signals são comumente utilizados para a sincronização de threads
- Cada thread CMSIS RTOS pode possuir até 32 signal flags

Signal Flags no RTOS

- Os Signal Flags não são "**criadas**" cada thread já possui 32 SFs em uma word de 32-bits

- As SFs são enviadas à uma determinada **thread** pelo **thread_id**

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

- #define **osFeature_Signals** 8
  - numero máximo de SF por thread

- Existem 3 funções para manipular os SFs no CMSIS RTOS
  - **osSignalSet** = Ativa flags sinalizadores para uma tarefa
  - **osSignalClear** = Desativa flags sinalizadores para uma tarefa
  - **osSignalWait** = Suspende execução até que flags sinalizadores específicos sejam ativados

Signal Flags no RTOS

int32_t **osSignalSet** (osThreadId **thread_id**, int32_t **signals**)

- Retorna a SF anterior da thread ou 0x80000000 para parâmetros incorretos

Parâmetros

- **thread_id** = identificador da thread obtido pelo osThreadCreate ou osThreadId
- **signals** = sinais para serem "**setados**" na thread

```
void Thread_2 (void const *arg);

osThreadDef (Thread_2, osPriorityHigh, 1, 0);

static void EX_Signal_1 (void)  {
  int32_t signals;
  uint32_t exec;
  osThreadId thread_id;

  thread_id = osThreadCreate (osThread(Thread_2), NULL);
  if (thread_id == NULL)  {
    // Failed to create a thread.
  }
  else  {
    signals = osSignalSet (thread_id, 0x00000005);         // Send signals to the created thread
  }
}
```

Signal Flags no RTOS

int32_t **osSignalClear** (osThreadId **thread_id**, int32_t **signals**)

- Retorna a SF anterior da thread ou 0x80000000 para parâmetros incorretos

Parâmetros

- **thread_id** = identificador da thread obtido pelo osThreadCreate ou osThreadId
- **signals** = sinais para serem "**limpos**" na thread

```
void Thread_2 (void const *arg);
osThreadDef (Thread_2, osPriorityHigh, 1, 0);

static void EX_Signal_1 (void)  {
  int32_t  signals;
  osThreadId thread_id;

  thread_id = osThreadCreate (osThread(Thread_2), NULL);
  if (thread_id == NULL)  {
    // Failed to create a thread.
  }
  else  {
f    :
    signals = osSignalClear (thread_id, 0x01);
  }
}
```

Signal Flags no RTOS

osEvent **osSignalWait** (int32_t **signals,** uint32_t **millisec**)

- Suspende a execução da thread em **"running"** até que um conjunto de SFs específico sejam **"setados"**
  - Quando SF=0 a thread é suspensa até que SF=1
  - Quando SF=1 a thread continua em execução
  - Quando a thread está em waiting seus SFs automaticamente são zerados
- Retorna a informação de um evento SF ou código de erro

Parâmetros

- **signals** = lista de SF que serão considerados eventos ou 0 para todos os SFs
- **millisec** = tempo máximo de espera em millisegundos

Retornos

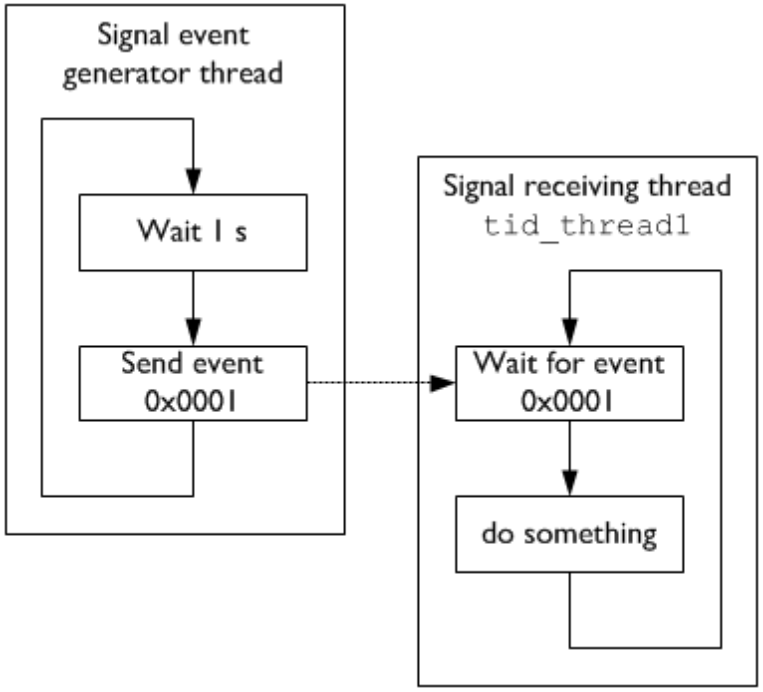
- osOK = não ocorreu o evento SF e o timeout é 0
- osEventTimeout = ocorreu timeout
- osEventSignal = ocorreu um evento de SF e elas foram zeradas
- osErrorValue = o parâmetro signals está incorreto (fora do range)
- osErrorISR = não pode ser chamado dentro de um ISR (interrupt service routine)

Signal Flags no RTOS

osEvent **osSignalWait** (int32_t **signals,** uint32_t **millisec**)

```
void Thread_2 (void const *arg);

osThreadDef (Thread_2, osPriorityHigh, 1, 0);

static void EX_Signal_1 (void)  {
  osThreadId thread_id;
  osEvent evt;

  thread_id = osThreadCreate (osThread(Thread_2), NULL);
  if (thread_id == NULL)  {
    // Failed to create a thread.
  }
  else  {
    :
    // wait for a signal
    evt = osSignalWait (0x01, 100);
    if (evt.status == osEventSignal)  {
          // handle event status
    }
  }
}
```

Comunicação por eventos SF



Simple signal event communication

1. In the thread (for example thread ID `tid_thread1`) that is supposed to wait for a signal, call the wait function:

```
osSignalWait (0x0001, osWaitForever); // wait forever for the signal 0x0001
```

2. In another thread (or threads) that are supposed to wake the waiting thread up call:

```
osSignalSet (tid_thread1, 0x0001);    // set the signal 0x0001 for thread tid_thread1
osDelay (1000);                       // wait for 1 second
```

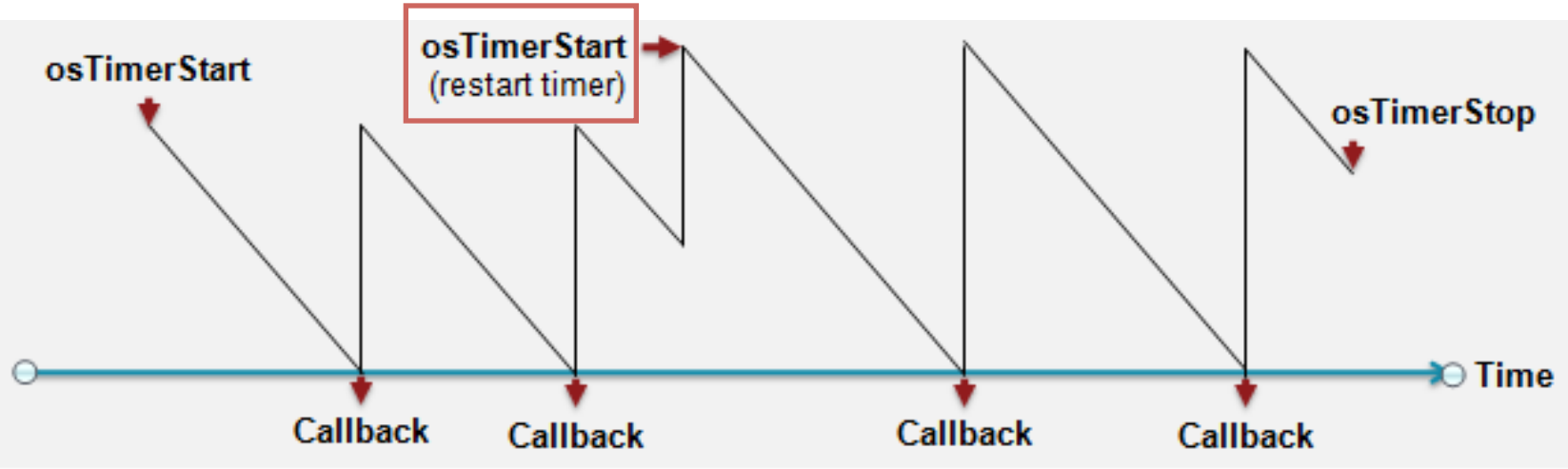
Exemplo de comunicação por SF no LPC1343

```
void led_thread(void const *args) {
    while (1) {
        // Signal flags that are reported as event are automatically cleared.
        osSignalWait(0x1, osWaitForever);
         pca_toggle(0);
    }
}
osThreadDef(led_thread, osPriorityNormal, 1,0);

int main (void) {

    osKernelInitialize();

    I2CInit( (uint32_t)I2CMASTER, 0 );

    osThreadId tid = osThreadCreate(osThread(led_thread), NULL);

    osKernelStart();

    while (1) {
        osDelay(1000);
        osSignalSet(tid, 0x1);
    }
}
```

Temporizadores

- O CMSIS RTOS permite a criação de temporizadores (**timers**) para a geração de eventos periódicos ou atrasos
- É gerada uma interrupção (**callback**) de estouro do temporizador
- Existem dois tipos de temporizadores
  - osTimerOnce = temporizador sem auto-reload (**one-shot**)
  - osTimerPeriodic = temporizador com auto-reload

Temporizador Periódico



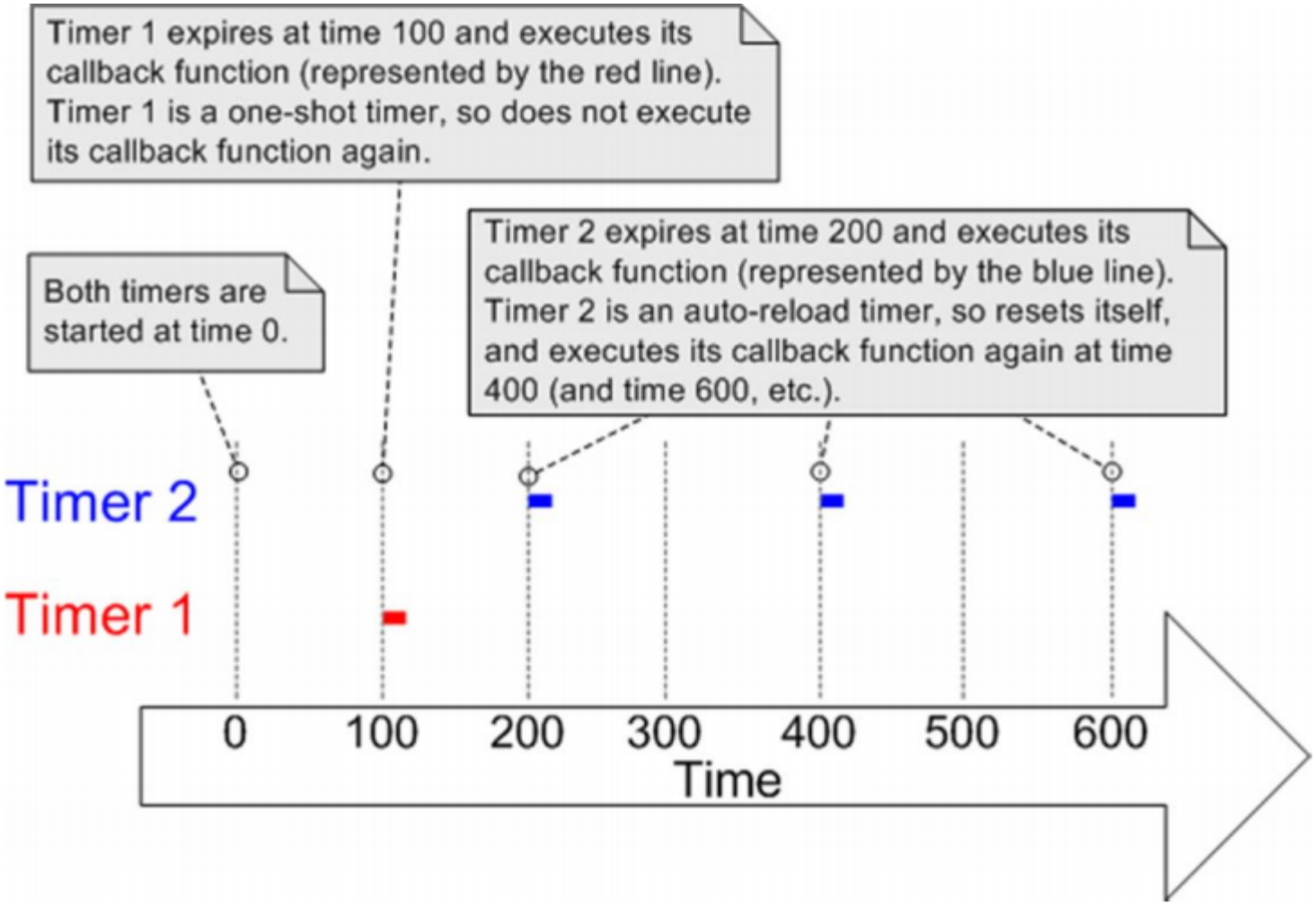
Behavior of a Periodic Timer

- **osTimerStart** define o valor inicial e inicia contagem regressiva
- **osTimerStop** para a contagem regressiva

- Quando a contagem atinge zero, é chamada a função de **callback**

Obs: para redefinir o valor inicial e reiniciar a contagem, é necessário executar osTimerStop antes de executar osTimerStart!

One-shot & Auto-reload



`osTimerOnce` *versus* `osTimerPeriodic`

Gerenciamento de Temporizadores

- **osTimerCreate**
  - Define os atributos da função de callback de um temporizador
- **osTimerStart**
  - Ativa um temporizador com um valor de tempo
- **osTimerStop**
  - Desativa um temporizador

Gerenciamento de Temporizadores

osTimerId **osTimerCreate** (osTimerDef_t * **timer_def**, os_timer_type **type**, void * **argument**)

- Cria um timer e associa uma função de **Callback**
- Retorna o identificador do Timer ou NULL para erro

Parâmetros

- **timer_def** = timer object referenced with osTimer.
- **type** = osTimerOnce for one-shot or osTimerPeriodic for periodic behavior.
- **argument** = argument to the timer call back function.

Gerenciamento de Temporizadores

osStatus **osTimerStart** (osTimerId **timer_id**, uint32_t **millisec**)

- Inicia e reinicia o timer
- Retorna o status

Parâmetros

- **timer_id** = identificador do timer, obtido com o osTimerCreate
- **millisec** = tempo de estouro do timer

Retornos

- osOK = o timer foi iniciado/reiniciado
- osErrorParameter = timer_id incorreto

Gerenciamento de Temporizadores

osStatus **osTimerStop** (osTimerId **timer_id**)

- Para o timer
- Retorna o status

Parâmetros

- **timer_id** = identificador do timer, obtido com o osTimerCreate

Retornos

- osOK = o timer foi parado
- osErrorParameter = timer_id incorreto
- osErrorResource = o timer não estava iniciado

Exemplo do uso de timer LPC1343

```
void blink(void const *n) {
    pca_toggle((int)n);
}

osTimerDef(blink_0, blink);
osTimerDef(blink_1, blink);
osTimerDef(blink_2, blink);
osTimerDef(blink_3, blink);

int main(void) {

    osKernelInitialize();

    SystemInit();
    I2CInit( (uint32_t)I2CMASTER, 0 );

    osTimerId timer_0 = osTimerCreate(osTimer(blink_0), osTimerPeriodic, (void *)0);
    osTimerId timer_1 = osTimerCreate(osTimer(blink_1), osTimerPeriodic, (void *)1);
    osTimerId timer_2 = osTimerCreate(osTimer(blink_2), osTimerPeriodic, (void *)2);
    osTimerId timer_3 = osTimerCreate(osTimer(blink_3), osTimerPeriodic, (void *)3);

    osTimerStart(timer_0, 2000);
    osTimerStart(timer_1, 1000);
    osTimerStart(timer_2,  500);
    osTimerStart(timer_3,  250);

    osKernelStart();
    osDelay(osWaitForever);
}
```

Chamada de ISR

- Apenas algumas funções específicas do CMSIS RTOS podem ser chamadas de Interrupções (ISR - Interrupt Service Routines)
  - osKernelRunning
  - osSignalSet
  - osSemaphoreRelease
  - osPoolAlloc, osPoolCAlloc, osPoolFree
  - osMessagePut, osMessageGet
  - osMailAlloc, osMailCAlloc, osMailGet, osMailPut, osMailFree
- As funções que não podem ser chamadas de ISR, irão verificar o status de interrupção e caso estejam em uma interrupção, vão gerar um código osErrorISR.