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

SISTEMAS EMBARCADOS

Comunicação entre threads

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

Comunicação entre threads

- É a passagem de informações (dados) entre duas ou mais threads

- O CMSIS RTOS fornece alguns métodos de comunicação entre threads
  - Blocos de memória (memory pool)
  - Fila de mensagens (message queue)
  - Fila de correspondências (mail queue)

Bloco de memória

- Consiste na definição de blocos fixos de memória
- Esse tipo de operação é mais rápida que **heap dinâmico** e não sofre com fragmentação
- Os blocos de memória podem ser acessados por threads e ISRs

Blocos de memória

- É possível criar blocos de memória compartilhados entre threads para troca de informações

- Essa modalidade de comunicação permite o uso de informações mais complexas **(não apenas inteiros e ponteiros)**

* Importante: a comunicação por blocos de memória é exclusiva para blocos com tamanho fixo

Blocos de memória

- **osPoolCreate**
  - Define e inicializa um pool de memória de tamanho fixo
- **osPoolAlloc**
  - Aloca bloco de memória
- **osPoolCAlloc**
  - Aloca bloco de memória e o inicializa com zeros
- **osPoolFree**
  - Devolve bloco de memória a um pool de memória

Blocos de memória

osPoolId **osPoolCreate** (const osPoolDef_t * **pool_def**)

- cria e inicializa os blocos de memória

Parâmetros

- **pool_def =** definição do bloco de memória

Retornos

- **retorna o ID do bloco de memória ou NULL para erro**

```
#include "cmsis_os.h"

typedef struct {
  uint8_t Buf[32];
  uint8_t Idx;
} MEM_BLOCK;

osPoolDef (MemPool, 8, MEM_BLOCK);

void CreateMemoryPool (void)  {
osPoolId MemPool_Id;

  MemPool_Id = osPoolCreate (osPool (MemPool));
  if (MemPool_Id != NULL)  {
    // memory pool created
  }
}
```

Blocos de memória

void* **osPoolAlloc** (osPoolId **pool_id**)

- Aloca um bloco de memória em um elemento Memory Pool

Parâmetros

- **pool_id =** identificador do bloco de memória

Retornos

- **retorna o endereço da memória reservada ou NULL para erro**

```
#include "cmsis_os.h"

typedef struct {
  uint8_t Buf[32];
  uint8_t Idx;
} MEM_BLOCK;

osPoolDef (MemPool, 8, MEM_BLOCK);

void AlocMemoryPoolBlock (void)  {
  osPoolId   MemPool_Id;
  MEM_BLOCK *addr;

  MemPool_Id = osPoolCreate (osPool (MemPool));
  if (MemPool_Id != NULL)  {
    :
    // allocate a memory block
    addr = (MEM_BLOCK *)osPoolAlloc (MemPool_Id);

    if (addr != NULL) {
      // memory block was allocated
      :
    }
  }
}
```

Blocos de memória

void* **osPoolCAlloc** (osPoolId **pool_id**)

- Aloca um bloco de memória em um elemento Memory Pool.
- **O bloco é iniciado com zeros**

Parâmetros

- **pool_id =** identificador do bloco de memória

Retornos

- **retorna o endereço da memória reservada ou NULL para erro**

```
#include "cmsis_os.h"

typedef struct {
  uint8_t Buf[32];
  uint8_t Idx;
} MEM_BLOCK;

osPoolDef (MemPool, 8, MEM_BLOCK);

void CAlocMemoryPoolBlock (void)  {
  osPoolId   MemPool_Id;
  MEM_BLOCK *addr;

  MemPool_Id = osPoolCreate (osPool (MemPool));
  if (MemPool_Id != NULL)  {
    :
    // allocate a memory block
    addr = (MEM_BLOCK *)osPoolCAlloc (MemPool_Id);

    if (addr != NULL) {
      // memory block was allocated
      :
    }
  }
}
```

Blocos de memória

osStatus **osPoolFree** (osPoolId **pool_id**,
void ***block**)

- libera um blocos de memória

Parâmetros

- **pool_id =** identificador do bloco de memória

Retornos

- osOK = o bloco de memória foi liberado
- osErrorValue = o bloco não faz parte do bloco de memória
- osErrorParameter = algum parâmetro está incorreto

```
#include "cmsis_os.h"

typedef struct {
  uint8_t Buf[32];
  uint8_t Idx;
} MEM_BLOCK;

osPoolDef (MemPool, 8, MEM_BLOCK);

void CAlocMemoryPoolBlock (void)  {
  osPoolId   MemPool_Id;
  MEM_BLOCK *addr;
  osStatus   status;

  MemPool_Id = osPoolCreate (osPool (MemPool));
  if (MemPool_Id != NULL)  {
    addr = (MEM_BLOCK *)osPoolCAlloc (MemPool_Id);
    if (addr != NULL) {
      :
      // return a memory block back to pool
      status = osPoolFree (MemPool_Id, addr);
      if (status==osOK)  {
        // handle status code
      }
    }
  }
}
```

Trabalhando com blocos de memória

1. Declare a data structure that combines a number of elements:

```
typedef struct {
  uint32_t length;
  uint32_t width;
  uint32_t height;
  uint32_t weight;
} properties_t;
```

2. Declare a memory pool of these objects as a block of memory:

```
osPoolDef (object_pool, 10, properties_t);  // Declare memory pool
osPoolId  (object_pool_id);                 // Memory pool ID
```

3. Then, create the memory pool in a thread:

```
object_pool_id = osPoolCreate(osPool(object_pool));
```

4. Allocate the pool within a thread and fill it with data:

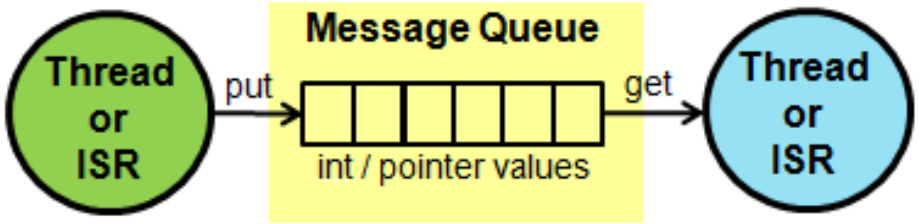
```
properties_t *object_data;
 object_data = (properties_t *) osPoolAlloc(object_pool_id);

object_data->length = 100;
object_data->width  = 10;
object_data->height = 23;
object_data->weight = 1000;
```

Fila de mensagens

- Na passagem de mensagens: uma thread envia o dado explicitamente enquanto outra thread recebe

- Esta informação funciona similar à uma I/O mas sem acesso direto a informação

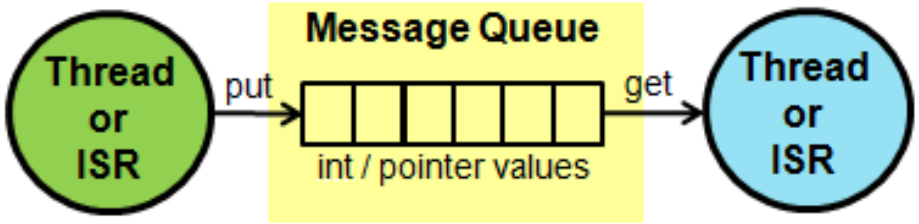
- A passagem de informações segue o formalismo de uma fila (FIFO - First In First Out)



Fila de mensagens

- São permitidos apenas dois tipos de mensagens
  - inteiros de 32 bits
  - ponteiros

- A manipulação da fila de mensagens é feita por meio de eventos e dos comandos de Put e Send



Gerenciamento da fila de mensagens

- **`osMessageCreate`**
  - Define e inicializa uma fila de mensagens
- **`osMessagePut`**
  - Coloca mensagem em uma fila de mensagens
- **`osMessageGet`**
  - Obtém mensagem de uma fila ou suspende a execução de uma tarefa até que uma mensagem chegue

Gerenciamento da fila de mensagens

osMessageQDef (**name, queue_sz, type**)

- **name =** nome da fila
- **queue_sz =** numero máximo de mensagens na fila
- **type =** tipo do dado da mensagem

osMessageQId **osMessageCreate** (const osMessageQDef_t * **queue_def**, osThreadId **thread_id**)

- **queue_def =** definição da fila de mensagens
- **thread_id =** identificador da thread

```
typedef struct {                                    // Message object structure
  float    voltage;                                 // AD result of measured voltage
  float    current;                                 // AD result of measured current
  int      counter;                                 // A counter value
} T_MEAS;

osPoolDef(mpool, 16, T_MEAS);                       // Define memory pool
osPoolId  mpool;
osMessageQDef(MsgBox, 16, &T_MEAS);                 // Define message queue
osMessageQId  MsgBox;
```

```
void StartApplication (void) {
  mpool = osPoolCreate(osPool(mpool));                    // create memory pool
  MsgBox = osMessageCreate(osMessageQ(MsgBox), NULL);     // create msg queue
```

Gerenciamento da fila de mensagens

osEvent **osMessageGet** (osMessageQId **queue_id**, uint32_t **millisec**)

- Suspende a execução da thread em execução (**running**) até que seja recebida uma mensagem.
- Se já existir uma mensagem disponível na fila, a função retorna imediatamente com a informação

Parâmetros

- **queue_id** = identificador da fila de mensagens
- **milisec** = tempo máximo de espera por uma mensagem ou 0 para no timeout
  - **quando milisec = 0** a função retorna instantaneamente
  - **quando milisec = osWaitForever** a função espera até a chegada de mensagens

Retornos

- osOK = não existe mensagem na fila e o timeout não foi especificado
- osEventTimeout = ocorreu timeout
- osEventMessage = foi recebida uma mensagem
- osErrorParameter = algum parâmetro está incorreto

Gerenciamento da fila de mensagens

osEvent **osMessageGet** (osMessageQId **queue_id**, uint32_t **millisec**)

```
//
//  Thread 2: Receive thread
//
void recv_thread (void const *argument) {
  T_MEAS  *rptr;
  osEvent  evt;

  for (;;) {
    evt = osMessageGet(MsgBox, osWaitForever);  // wait for message
    if (evt.status == osEventMessage) {
      rptr = evt.value.p;
      printf ("\nVoltage: %.2f V\n", rptr->voltage);
      printf ("Current: %.2f A\n", rptr->current);
      printf ("Number of cycles: %d\n", rptr->counter);
```

Gerenciamento da fila de mensagens

osStatus **osMessagePut** (osMessageQId **queue_id**, uint32_t **info**, uint32_t **millisec**)

- Insere uma mensagem dentro de uma fila de mensagens

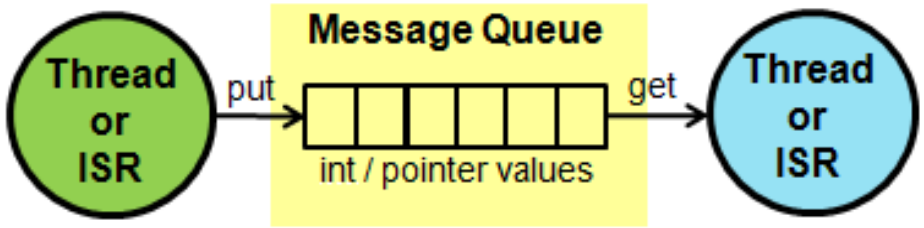
Parâmetros

- **queue_id** = identificador da fila de mensagens
- **info** = informação da mensagem
- **milisec** = tempo máximo de espera por uma mensagem ou 0 para no timeout
  - **quando milisec = 0** a função retorna instantaneamente
  - **quando milisec = osWaitForever** a função espera até a chegada de mensagens

Retornos

- osOK = a mensagem foi inserida na fila
- osErrorResource = não tem memória disponível na fila
- osErrorTimeoutResource = não houve espaço disponível na fila dentro do tempo máximo estipulado
- osErrorParameter = algum parâmetro está incorreto

Trabalhando com a fila de mensagens



1. Setup the message queue:

```
osMessageQDef(message_q, 5, uint32_t); // Declare a message queue
osMessageQId (message_q_id);           // Declare an ID for the message queue
```

2. Then, create the message queue in a thread:

```
message_q_id = osMessageCreate(osMessageQ(message_q), NULL);
```

3. Fill the message queue with data:

```
uint32_t data = 512;

osMailPut(message_q_id, data, osWaitForever);
```

4. From the receiving thread access the data using:

```
osEvent event = osMessageGet(message_q_id, osWaitForever);
```

Exemplo com Blocos de Memória e Fila de Mensagens

```
typedef struct {
    int32_t     temp;           /* AD result of measured temperature      */
    uint32_t    light;          /* AD result of measured light            */
    uint32_t    trimpot;        /* AD result of measured trimpot voltage  */
    uint32_t    counter;        /* message counter                        */
} message_t;

osPoolDef(mpool, 16, message_t);
osPoolId  mpool;

osMessageQDef(queue, 16, message_t);
osMessageQId  queue;

void send_thread (void const *args) {
    uint32_t i = 0;
    while (1) {
        i++;
        message_t *message = (message_t*)osPoolAlloc(mpool);
        message->temp = get_temperature();
        message->light = get_light();
        message->trimpot = get_trimpot();
        message->counter = i;
        osMessagePut(queue, (uint32_t)message, osWaitForever);
        osDelay(1000);
    }
}
osThreadDef(send_thread, osPriorityNormal, 1,0);
```

Exemplo com Blocos de Memória e Fila de Mensagens

```
osThreadDef(send_thread, osPriorityNormal, 1,0);

void receive_thread (void const *args) {
     while (1) {
        osEvent evt = osMessageGet(queue, osWaitForever);
        if (evt.status == osEventMessage) {
            message_t *message = (message_t*)evt.value.p;
            printf("\nTemperature: %i \n\r", message->temp);
            printf("Light: %u \n\r"     , message->light);
            printf("Trimpot: %u \n\r"     , message->trimpot);
            printf("Number of messages: %u\n\r", message->counter);

            osPoolFree(mpool, message);
        }
    }
}
osThreadDef(receive_thread, osPriorityNormal, 1,0);
```

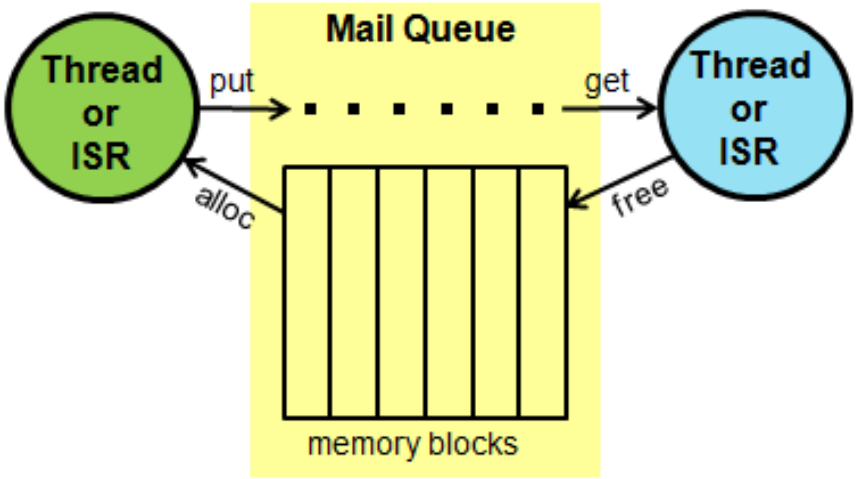
```
int main (void) {

    osKernelInitialize();

    I2CInit( (uint32_t)I2CMASTER, 0 );
    ADCInit( ADC_CLK );

    light_init();
    acc_init();
    temp_init (&getTicks);

    mpool = osPoolCreate(osPool(mpool));
    queue = osMessageCreate(osMessageQ(queue), NULL);

    osThreadCreate(osThread(send_thread), NULL);
    osThreadCreate(osThread(receive_thread), NULL);

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

Fila de Correspondências

- É similar à Fila de Mensagens porém os dados que serão transferidos consistem em **blocos de memória que precisam ser alocados e liberados**

- Emprega a Fila de Mensagens para criar **blocos de memórias formatados** e repassar os ponteiros para esses blocos por uma Fila de Mensagens

- **Apenas um ponteiro 32 bits** é utilizado para comunicar blocos de memória entre threads



Gerenciamento da Fila de Correspondências

- **`osMailCreate`**
  - Define e inicializa uma fila de correspondências com blocos de memória de tamanho fixo
- **`osMailAlloc`**
  - Aloca bloco de memória
- **`osMailCAlloc`**
  - Aloca bloco de memória e o inicializa com zeros
- **`osMailPut`**
  - Coloca bloco de memória em uma fila de correspondências
- **`osMailGet`**
  - Obtém correspondêncis ou suspende a execução de uma tarefa até que uma correspondência chegue
- **`osMailFree`**
  - Devolve bloco de memória a uma fila de correspondências

Gerenciamento da Fila de Correspondências

osMailQDef (**name, queue_sz, type**)

- **name =** nome da fila de correspondências
- **queue_sz =** numero máximo de correspondências na fila
- **type =** tipo do dado de um elemento simples

osMailQId **osMailCreate** (const osMailQDef_t * **queue_def**, osThreadId **thread_id**)

- **queue_def =** definição da fila de mensagens
- **thread_id =** identificador da thread

```
typedef struct {                                  // Mail object structure
  float    voltage;                               // AD result of measured voltage
  float    current;                               // AD result of measured current
  int      counter;                               // A counter value
} T_MEAS;

osMailQDef(mail, 16, T_MEAS);                     // Define mail queue
osMailQId  mail;

void send_thread (void const *argument);          // forward reference
void recv_thread (void const *argument);

osThreadDef(send_thread, osPriorityNormal, 1, 0);     // thread definitions
osThreadDef(recv_thread, osPriorityNormal, 1, 2000);
```

```
void StartApplication (void) {
  mail = osMailCreate(osMailQ(mail), NULL);      // create mail queue

  tid_thread1 = osThreadCreate(osThread(send_thread), NULL);
  tid_thread2 = osThreadCreate(osThread(recv_thread), NULL);
  :
}
```

Gerenciamento da Fila de Correspondências

void * **osMailAlloc** (osMailQId queue_id, uint32_t millisec)

void * **osMailCAlloc** (osMailQId queue_id, uint32_t millisec)

- Aloca um bloco de memória na fila de correspondências
- CAlloc inicia o bloco com zeros
- Pode ser chamada de ISRs

Parâmetros

- **queue_id** = identificador da fila de correspondências
- **millisec** = tempo máximo de espera para um espaço (slot) de memória da fila esteja disponível em milisegundos. Pode-se utilizar **osWaitForever** para esperar indefinidamente

```
typedef struct {                                  // Mail object structure
  float    voltage;                               // AD result of measured voltage
  float    current;                               // AD result of measured current
  int      counter;                               // A counter value
} T_MEAS;

osMailQDef(mail, 16, T_MEAS);                     // Define mail queue
osMailQId  mail;

void send_thread (void const *argument);          // forward reference
void recv_thread (void const *argument);

osThreadDef(send_thread, osPriorityNormal, 1, 0);     // thread definitions
osThreadDef(recv_thread, osPriorityNormal, 1, 2000);

//
//  Thread 1: Send thread
//
void send_thread (void const *argument) {
  T_MEAS *mptr;

  mptr = osMailAlloc(mail, osWaitForever);        // Allocate memory
  mptr->voltage = 223.72;                         // Set the mail content
  mptr->current = 17.54;
  mptr->counter = 120786;
```

Gerenciamento da Fila de Correspondências

osEvent **osMailGet** (osMailQId **queue_id**, uint32_t **millisec**)

- Suspende a execução da thread em execução (**running**) até que seja recebida uma correspondência
- Se já existir uma correspondência disponível na fila, a função retorna imediatamente com a informação
- Pode ser chamada de ISRs

Parâmetros

- **queue_id** = identificador da fila de correspondências
- **milisec** = tempo máximo de espera por uma correspondência ou 0 para no timeout
  - **quando milisec = 0** a função retorna instantaneamente
  - **quando milisec = osWaitForever** a função espera até a chegada de correspondências

Retornos

- osOK = não existe correspondência na fila e o timeout não foi especificado
- osEventTimeout = ocorreu timeout
- osEventMail = foi recebida uma correspondência
- osErrorParameter = algum parâmetro está incorreto

Gerenciamento da Fila de Correspondências

osStatus **osMailPut** (osMailQId **queue_id**, void * **mail**)

- Coloca um bloco de memória com correspondência dentro de uma fila de correspondências

Parâmetros

- **queue_id** = identificador da fila de correspondências
- **mail** = bloco de memória alocado com **osMailAlloc** ou **osMailCAlloc**

Retornos

- osOK = a correspondência foi inserida na fila
- osErrorValue = a correspondência não foi alocada em um slot de memória
- osErrorParameter = algum parâmetro está incorreto

Gerenciamento da Fila de Correspondências

osStatus **osMailFree** (osMailQId **queue_id**, void * **mail**)

- Libera um bloco de memória utilizado por uma correspondência

Parâmetros

- **queue_id** = identificador da fila de correspondências
- **mail** = ponteiro para o bloco de memória obtido pelo **osMailGet**

Retornos

- osOK = o bloco de memória foi liberado
- osErrorValue = o bloco de memória não faz parte da fila de correspondências
- osErrorParameter = algum parâmetro está incorreto

Trabalhando com Fila de Correspondências

1. Declare a data structure that combines a number of elements:

```
typedef struct {
  uint32_t length;
  uint32_t width;
  uint32_t height;
  uint32_t weight;
} properties_t;
```

2. Declare a mail queue made up of these objects:

```
osMailQDef (object_pool_q, 10, properties_t);  // Declare mail queue
osMailQId  (object_pool_q_id);                 // Mail queue ID
```

3. Then, create the mail pool in a thread:

```
object_pool_q_id = osMailCreate(osMailQ(object_pool_q), NULL);
```

4. Allocate the mail queue within a thread and fill it with data:

```
properties_t *object_data;
 object_data = (properties_t *) osMailAlloc(object_pool_q_id, osWaitForever);

object_data->length = 100;
object_data->width = 10;
object_data->height = 23;
object_data->weight = 1000;
```

5. Pass the pointer to the mail queue to another thread:

```
osMailPut(object_pool_q_id, object_data);
```

6. Access the data in another thread:

```
osEvent event = osMailGet(properties_q_id, osWaitForever);
properties_t *received = (properties_t *)event.value.p;       // ".p" indicates that the message is a pointer
my_length(received->length);
```

7. Once the data has been used, the memory block must be freed so that the memory pool can be reused

```
osMailFree(object_pool_q_id, received);
```

Exemplo com Fila de Correspondências

```
typedef struct {
    int32_t     temp;           /* AD result of measured temperature      */
    uint32_t    light;          /* AD result of measured light            */
    uint32_t    trimpot;        /* AD result of measured trimpot voltage  */
    uint32_t    counter;        /* message counter                        */
} mail_t;

osMailQDef(mail_box, 16, mail_t);
osMailQId  mail_box;

void send_thread (void const *args) {
    uint32_t i = 0;
    while (1) {
        i++;
        mail_t *mail = (mail_t*)osMailAlloc(mail_box, osWaitForever);
        mail->temp = get_temperature();
        mail->light = get_light();
        mail->trimpot = get_trimpot();
        mail->counter = i;
        osMailPut(mail_box, mail);
        osDelay(1000);
    }
}
osThreadDef(send_thread, osPriorityNormal, 1, 0);
```

Exemplo com Fila de Correspondências

```
void receive_thread (void const *args) {
     while (1) {
        osEvent evt = osMailGet(mail_box, osWaitForever);
        if (evt.status == osEventMail) {
            mail_t *mail = (mail_t*)evt.value.p;
            printf("\nTemperature: %i \n\r", mail->temp);
            printf("Light: %u \n\r" , mail->light);
            printf("Trimpot: %u \n\r" , mail->trimpot);
            printf("Number of messages: %u\n\r", mail->counter);

            osMailFree(mail_box, mail);
        }
    }
}
osThreadDef(receive_thread, osPriorityNormal, 1,0);
```

```
int main (void) {
    osKernelInitialize();

    I2CInit( (uint32_t)I2CMASTER, 0 );
    ADCInit( ADC_CLK );

    light_init();
    acc_init();
    temp_init (&getTicks);

    mail_box = osMailCreate(osMailQ(mail_box), NULL);

    osThreadCreate(osThread(send_thread), NULL);
    osThreadCreate(osThread(receive_thread), NULL);

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

Tabela Resumo – Ciclo de Vida

	Thread	Timer	Mutex	Semaphore	Memory Pool	Message Queue	Mail Queue
Definição	• osThread Def	• osTimer Def	• osMutex Def	• osSemaphore Def	• osPool Def	• osMessageQ Def	• osMailQ Def
Acesso	• osThread	• osTimer	• osMutex	• osSemaphore	• osPool	• osMessageQ	• osMailQ
Inicialização	• osThread Create	• osTimer Create	• osMutex Create	• osSemaphore Create	• osPool Create	• osMessage Create	• osMail Create
Operação	• osThread Yield • osThread GetId • osThread SetPriority • osThread GetPriority	• osTimer Start • osTimer Stop	• osMutex Wait • osMutex Release	• osSemaphore Wait • osSemaphore Release	• osPool Alloc • osPool CAlloc • osPool Free	• osMessage Put • osMessage Get	• osMail Alloc • osMail CAlloc • osMail Put • osMail Get • osMail Free
Término	• osThread Terminate	• osTimer Delete	• osMutex Delete	• osSemaphore Delete			