UTFPR

Microcontroladores: (LT36D)
Prof: DaLuz

UTFPR

REENTRÂNCIA

Definição:

- Funções **reentrantes** são aquelas que **podem** ser chamadas por **mais** de uma **tarefa** e ainda assim, **sempre** trabalham **concorrentemente**, mesmo que o **RTOS chaveie** de uma tarefa para outra no **meio** da execução da função.

UTFPR

REENTRÂNCIA

Regras:

- Uma função **reentrante** não pode usar variáveis de uma forma não atômica, a menos que elas estejam armazenadas na **pilha** da tarefa que chamou a função ou são variáveis **privadas** desta **pilha**.
- Uma função **reentrante** não pode chamar outra função que não seja ela própria uma função **reentrante**.
- Uma função **reentrante** não pode usar o ***hardware*** de uma forma não atômica.

UTFPR

REENTRÂNCIA

Regras:

- A melhor maneira de compreender a **reentrância** e em particular a primeira regra anterior, é entender onde os compiladores C armazenam as variáveis. Saber:

- **Pilha** x posição fixa de **memória**.

UTFPR

REENTRÂNCIA

Regras:

- **Static int** – é armazenada em uma localização física da memória e portanto é compartilhada por qualquer tarefa.
- **Public int** – A única diferença da static é que as funções de outros módulos podem acessar public int.
- **Var. Globais e Var. inicializada** - uma variável está na memória, pode estar na pilha para variáveis locais ...
- **Ponteiro** – localização fixa na memória, portanto compartilhada ... Se uma função alterar o valor ...
- **Parâmetros** – Estes são criados na pilha, portanto várias tarefas podem executar a chamada da função.
- **Ptr_parm** – Estes são criados na pilha, portanto em teoria podem ser reentrantes, mas se a função alterar o valor dele, temos que verificar onde isso se encontra, pois pode gerar problemas.
- **Local** – Está na pilha.

UTFPR

REENTRÂNCIA

Regras:

Não Reentrante:

```
int g_var = 1;

int f()
{
  g_var = g_var + 2;
  return (g_var);
}

int g()
{
  return (f() + 2);
}
```

Reentrante:

```
int f(int i)
{
  return (i + 2);
}

int g(int i)
{
  return (f(i) + 2);
}
```

PARALELISMO

Técnicas:

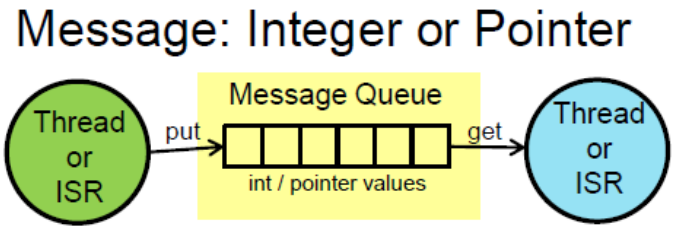
- **Comunicação** entre tarefas.
- **Sincronização** entre tarefas
- **Compartilhamento** de Recursos e Exclusão Mútua
- Técnicas de **Agendamento** / **Escalonamento** de tarefas

UTFPR

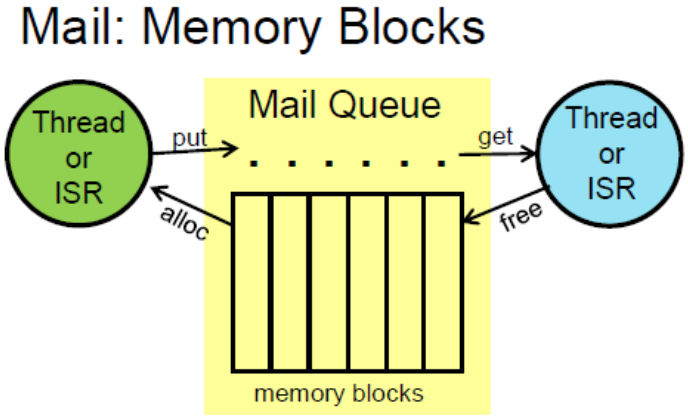
COMUNICAÇÃO

Entre tarefas:

- Fila de **Números** ou **Ponteiros**:



- Fila de **Mensagens**:

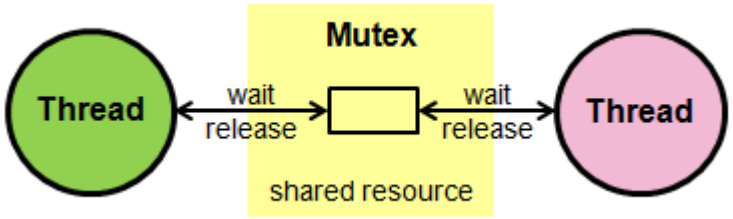


UTFPR

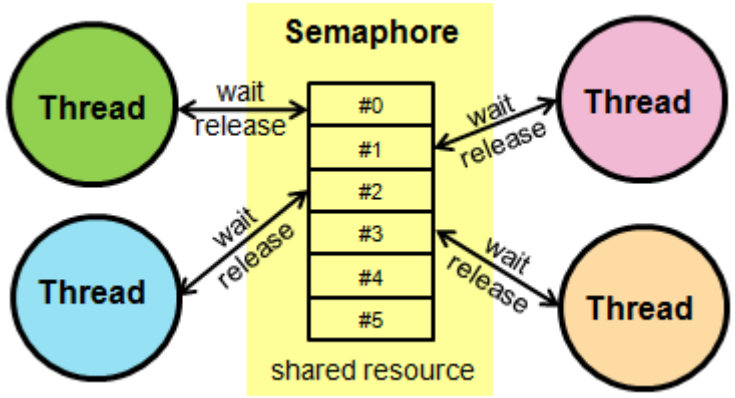
SINCRONIZAÇÃO

Entre tarefas:

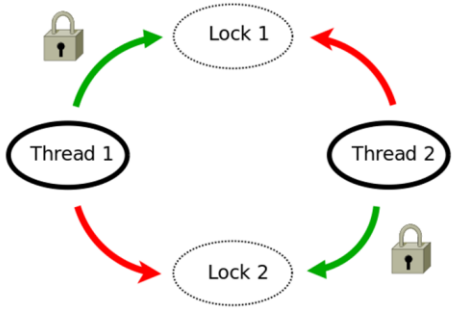
Mutex:



Semáforo:



Risco:

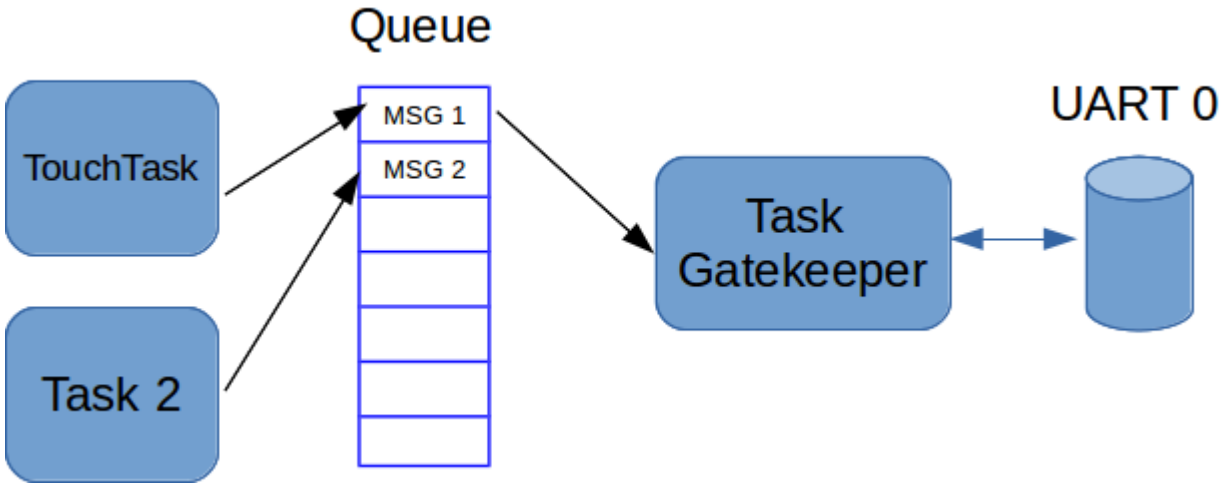


UTFPR

SINCRONIZAÇÃO

GateKeeper:

- Acesso ao **recurso** é efetuado por uma **única** tarefa.
- Tarefas que pretendam **acessar** o recurso comunicam com a ***Gatekeeper***, e.g. via mensagens.



UTFPR

ThreadX

Informações:

Arquivos:

FILENAME	DESCRIPTION
tx_api.h	C header file containing all system equates, data structures, and service prototypes.
tx_port.h	C header file containing all development-tool and targetspecific data definitions and structures.
demo_threadx.c	C file containing a small demo application.
tx.a (or tx.lib)	Binary version of the ThreadX C library that is distributed with the *standard* package.

Passos importantes para criar um projeto:

1) Inserir o arquivo **tx_api.h** no projeto.

2) Criar um **C** padrão e no **main()** deverá existir a chamada da função: **tx_kernel_enter()**.

3) Criar a **tx_application_define()**, aqui se define todas as estruturas do projeto ...

4) Inserir o "*Run-Time Library*" de **tx.lib** ou **tx.a** no projeto ...

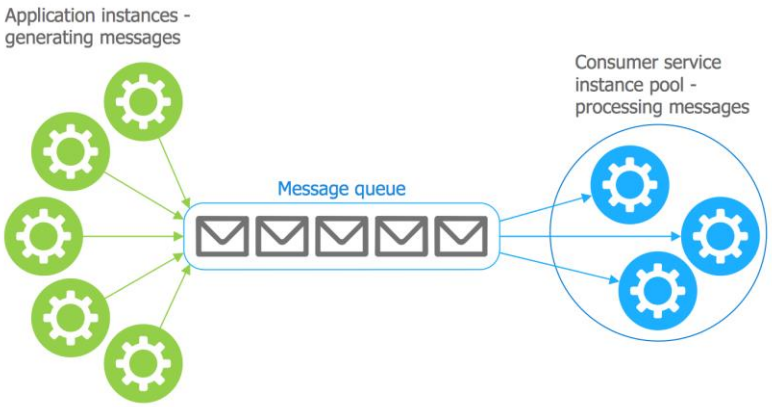
*Ref. **

UTFPR

ThreadX

Informações:

Mensagens:



tx_queue_create 180
Create message queue

tx_queue_delete 182
Delete message queue

tx_queue_flush 184
Empty messages in message queue

tx_queue_front_send 186
Send message to the front of queue

tx_queue_info_get 188
Retrieve information about queue

tx_queue_performance_info_get 190
Get queue performance information

tx_queue_performance_system_info_get 192
Get queue system performance information

tx_queue_prioritize 194
Prioritize queue suspension list

*Ref. **

UTFPR

ThreadX

Informações:

Exemplo:

```
TX_QUEUE queue_0;
static uint8_t queue_memory_queue_0[20];

UINT err_queue_0;

// UINT tx_queue_create(TX_QUEUE *queue_ptr, CHAR *name_ptr, UINT message_size, VOID *queue_start, ULONG queue_size);
err_queue_0 = tx_queue_create (&queue_0, (CHAR *) "queue 0", 1, &queue_memory_queue_0, sizeof(queue_memory_queue_0));
```

Example for a queue of size 20 bytes where each message is 1 word long, hence, the capacity is 5 messages.

queue control structure

queue storage structure

SSP code

Sender Task

```
UINT status;
ULONG thread_1_messages_sent;

/* Send message to queue 0.  */
// UINT tx_queue_send(TX_QUEUE *queue_ptr, VOID *source_ptr, ULONG wait_option);
status =  tx_queue_send(&queue_0, &thread_1_messages_sent, TX_WAIT_FOREVER);

/* Check completion status.  */
if (status != TX_SUCCESS)
    break;
```

Receiver Task

```
UINT status;
ULONG received_message;
while (1)
{

    /* Retrieve a message from the queue.  */
    // UINT tx_queue_receive(TX_QUEUE *queue_ptr, VOID *destination_ptr, ULONG wait_option);
    status = tx_queue_receive(&queue_0, &received_message, TX_WAIT_FOREVER);

    /* Check completion status and make sure the message is what we expected.  */
    if (status != TX_SUCCESS)
        break;
```

*Ref. **

UTFPR

ThreadX

Informações:

- **Protótipos**:

Mutex Services

```
UINT    tx_mutex_create(TX_MUTEX *mutex_ptr, CHAR *name_ptr,
           UINT inherit);
UINT    tx_mutex_delete(TX_MUTEX *mutex_ptr);
UINT    tx_mutex_get(TX_MUTEX *mutex_ptr, ULONG wait_option);
UINT    tx_mutex_prioritize(TX_MUTEX *mutex_ptr);
UINT    tx_mutex_put(TX_MUTEX *mutex_ptr);
```

- **Exemplo**:

```
TX_MUTEX my_mutex;
UINT status;
status = tx_mutex_create(&my_mutex,"my_mutex_name", TX_NO_INHERIT);
status = tx_mutex_get(&my_mutex, TX_WAIT_FOREVER);
```

*Ref. **

UTFPR

ThreadX

Informações:

Protótipos:

Mutex Services

```
UINT    tx_mutex_create(TX_MUTEX *mutex_ptr, CHAR *name_ptr,
           UINT inherit);
UINT    tx_mutex_delete(TX_MUTEX *mutex_ptr);
UINT    tx_mutex_get(TX_MUTEX *mutex_ptr, ULONG wait_option);
UINT    tx_mutex_prioritize(TX_MUTEX *mutex_ptr);
UINT    tx_mutex_put(TX_MUTEX *mutex_ptr);
```

Return Values

TX_SUCCESS	(0x00)	Successful mutex get operation.
TX_DELETED	(0x01)	Mutex was deleted while thread was suspended.
TX_NOT_AVAILABLE	(0x1D)	Service was unable to get ownership of the mutex within the specified time to wait.
TX_WAIT_ABORTED	(0x1A)	Suspension was aborted by another thread, timer, or ISR.
TX_MUTEX_ERROR	(0x1C)	Invalid mutex pointer.
TX_WAIT_ERROR	(0x04)	A wait option other than TX_NO_WAIT was specified on a call from a non-thread.
TX_CALLER_ERROR	(0x13)	Invalid caller of this service.

Exemplo:

```
TX_MUTEX my_mutex;
UINT status;
status = tx_mutex_create(&my_mutex,"my_mutex_name", TX_NO_INHERIT);
status = tx_mutex_get(&my_mutex, TX_WAIT_FOREVER);
```

*Ref. **

UTFPR

ThreadX

Informações:

Protótipos:

Thread Control Services

```
UINT      tx_thread_create(TX_THREAD *thread_ptr,
             CHAR *name_ptr,
             VOID (*entry_function)(ULONG), ULONG entry_input,
             VOID *stack_start, ULONG stack_size,
             UINT priority, UINT preempt_threshold,
             ULONG time_slice, UINT auto_start);

UINT      tx_thread_delete(TX_THREAD *thread_ptr);

UINT      tx_thread_entry_exit_notify(TX_THREAD *thread_ptr,
             VOID (*thread_entry_exit_notify)(TX_THREAD *, UINT));

TX_THREAD *tx_thread_identify(VOID);

UINT      tx_thread_info_get(TX_THREAD *thread_ptr, CHAR **name,
             UINT *state, ULONG *run_count, UINT *priority,
             UINT *preemption_threshold, ULONG *time_slice,
             TX_THREAD **next_thread,
             TX_THREAD **next_suspended_thread);

UINT      tx_thread_terminate(TX_THREAD *thread_ptr);

UINT      tx_thread_time_slice_change(TX_THREAD *thread_ptr,
             ULONG new_time_slice, ULONG *old_time_slice);

UINT      tx_thread_wait_abort(TX_THREAD *thread_ptr);
```

*Ref. **

UTFPR

ThreadX

Informações:

- **Protótipos**:

Time Services

```
ULONG   tx_time_get(VOID);
VOID    tx_time_set(ULONG new_time);
```

Timer Services

```
UINT    tx_timer_activate(TX_TIMER *timer_ptr);

UINT    tx_timer_change(TX_TIMER *timer_ptr,
           ULONG initial_ticks,
           ULONG reschedule_ticks);

UINT    tx_timer_create(TX_TIMER *timer_ptr,
           CHAR *name_ptr,
           VOID (*expiration_function)(ULONG),
           ULONG expiration_input, ULONG initial_ticks,
           ULONG reschedule_ticks, UINT auto_activate);

UINT    tx_timer_deactivate(TX_TIMER *timer_ptr);

UINT    tx_timer_delete(TX_TIMER *timer_ptr);

UINT    tx_timer_info_get(TX_TIMER *timer_ptr, CHAR **name,
           UINT *active, ULONG *remaining_ticks,
           ULONG *reschedule_ticks,
           TX_TIMER **next_timer);
```

*Ref. **

UTFPR

ThreadX

Informações:

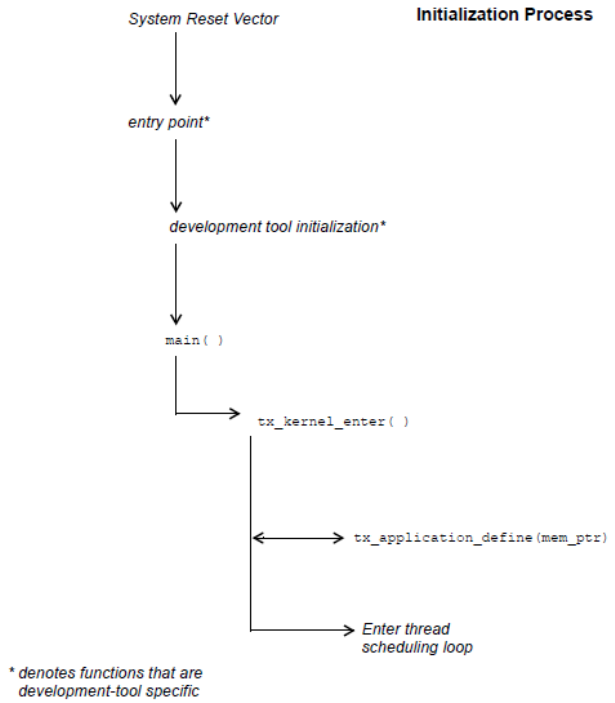


FIGURE 4. Initialization Process

```
#include           "tx_api.h"

unsigned long      my_thread_counter = 0;
TX_THREAD          my_thread;

main( )
{
      /* Enter the ThreadX kernel.  */
      tx_kernel_enter( );
}

void  tx_application_define(void *first_unused_memory)
{

      /* Create my_thread!  */
      tx_thread_create(&my_thread, "My Thread",
          my_thread_entry, 0x1234, first_unused_memory, 1024,
             3, 3, TX_NO_TIME_SLICE, TX_AUTO_START);
}

void   my_thread_entry(ULONG  thread_input)
{
      /* Enter into a forever loop.  */
      while(1)
      {

            /* Increment thread counter.  */
            my_thread_counter++;

            /* Sleep for 1 tick.  */
            tx_thread_sleep(1);
      }
}
```

FIGURE 1. Template for Application Development *Ref. **

UTFPR

ThreadX

Referências:

Documentação em português:
https://learn.microsoft.com/pt-br/azure/rtos/threadx/

Capítulo 3:
https://learn.microsoft.com/pt-br/azure/rtos/threadx/chapter3

Capítulo 4:
https://learn.microsoft.com/pt-br/azure/rtos/threadx/chapter4

Capítulo 6:
https://learn.microsoft.com/pt-br/azure/rtos/threadx/chapter6

UTFPR

Referências:

Continuação dos Labs ...
Lab 6:
http://www.elf74.daeln.com.br/Labs/Lab6_Tiva.pdf

* *Refs* ↔ Renesas.com, Pixabay.com, wikimedia.org, flickr, community.arm.com, Undergraduated course Renesas / CWS71-Prof. Douglas P. B. R. e Robson L., *ytchannel* Gustavo W. D., *ARMv7-M Architecture Reference Manual, CSW40-Sistemas Microcontrolados – Prof. Guilherme P., toshiba.semicon-storage.com, microntrollerslab.com, lfeletronics.com.br, elf74-Prof. Hugo V. N., stm. st.com, jblopen.com, microsoft.com.*