Interprocess Communication Mechanism and Keil RTX

SISTEMAS EMPOTRADOS

Keil RTX Real-Time Kernel

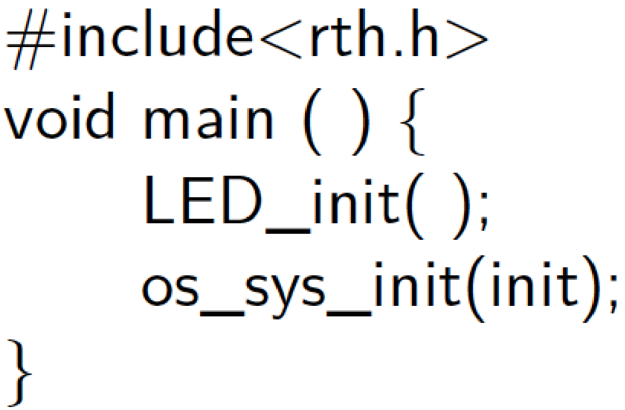
The Keil RTX is a royalty-free, RTOS targeted for micro controller applications.

The RTX is supported on all Cortex-M processors in addition to traditional ARM processors such as ARM7 and ARM9. It has the following features:

- Flexible scheduler, which supports preemptive, round-robin, and collaborative scheduling schemes.
- Support for mailboxes, events (up to 16 per tasks), semaphores, mutex, and timers.
- An unlimited number of defined tasks, with a maximum of 250 active tasks at a time.
- Up to 255 task priority levels.
- Support for multithreading and thread-safe operations
- Fast context switching time

OS Startup Stages

- The on_sys_init ( ) function initializes and start the OS. It must be called from mai ( ) and does not return



```
#include<rth.h>
void main ( ) {
        LED_init( );
        os_sys_init(init);
}
```

- To enable the RTX kernel option in Keil (Options for Target > Target)

Operating system : RTX Kernel

RTX Tasks

- Each taks must be declared with __task keyword, example:

```
__task void init (void) {
                t_blinky = os_tsk_create(blinky, 1);
                 os_tsk_delete_self ( );
}
```

- Task States in RTX Kernel:

State		Description
RUNNING		Currently running
READY		In the queue of tasks is ready to run.
WAITING	WAIT_DLY	For a delay
	WAIT_ITV	For a interval
	WAIT_OR	For at least one event
	WAIT_AND	For all a set of events
	WAIT_SEM	For a semaphore
	WAIT_MUT	For a mutex
	WAIT_MBX	For a mailbox message
INACTIVE		Task not started or deleted

RTX Tasks

- For each task, a task identifier value is requires (*OS_TID*). This taks ID value es assigned when the task is created and is required for intertask communications. *Os_tsk_self ( )*, return the taks ID.
- Functions to create new tasks:

Functions	Description
Os_tsk_create	Create a new task
Os_tsk_create_ex	Create a new task with an argument passing to the new task

- Examples:

```
   OS_TID t_blinky;
    t_blinky = os_tsk_create(blinky, 1);

   OS_TID  id2;
   ….
   id2 = os_task_create_ex(task1, p, &var);
  __task void task1 (int *argv){ …}
```

RTX Tasks

- Functions to delete tasks

Functions	Description
Os_tsk_delete	Delete a task
Os_tsk_delete_self	Delete the task itself

- Functions to manage task priority level

Functions	Description
Os_tsk_prio	Change the priority level of a task
Os_tsk_prio_self	Change tha priority level of a current task

- Examples:

```
os_task_prio(tid, p);     // tid=task id, new priority p
os_task_prio_self(p);   // current task new priority p
```

Mutual Exclusive

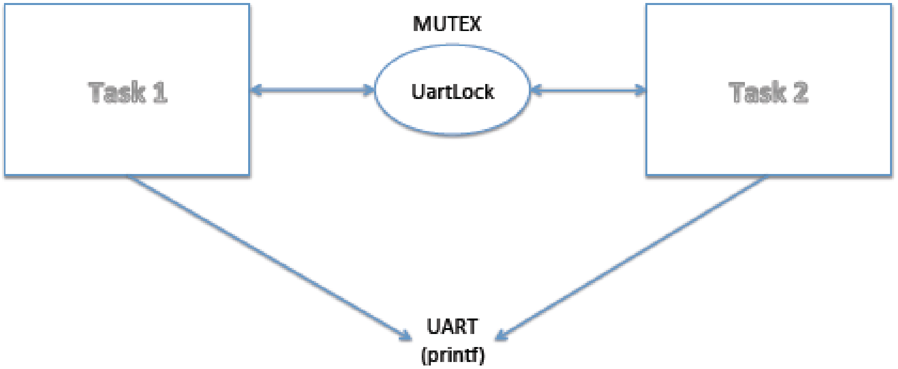
Critical regions

section of code that cannot be interrupted by another process.

Examples: writing shared memory; accessing I/O device.

Mutual exclusive

MUTEX to ensure that only a task can access to a hardware resource at one time



RTX Mutual Exclusion

OS_MUT ml; //declare MUTEX object ml

Os_mut_init (ml); //initialize object ml

Os_mut_wait(ml, timeout); //try to acquire mutex ml and return of:

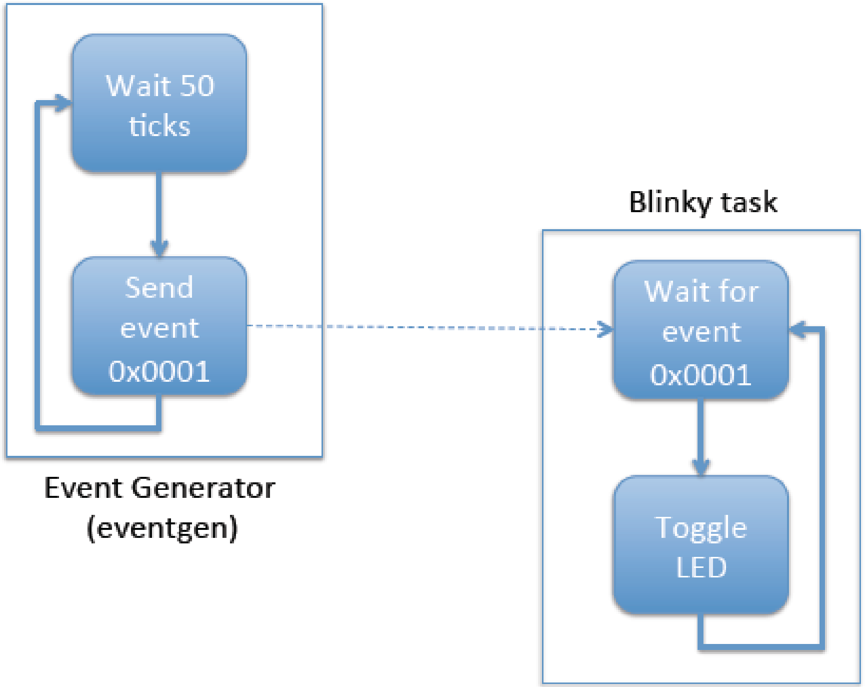
- ✓ OS_R_OK = acquired immediately (object was not locked)
- ✓ OS_R_MUT = acquired after a wait (object was locked)
- ✓ OS_R_TMO = not acquired after timeout

Os_mut_release(ml); // decrement counter to release object

Event Communications

A event is simple because it does not pass data beyond the existence of the event itself. A event is analogous to an interrupt, but it is entirely a software creation.

A event is generated by a process and transmitted to another process by the operating system.

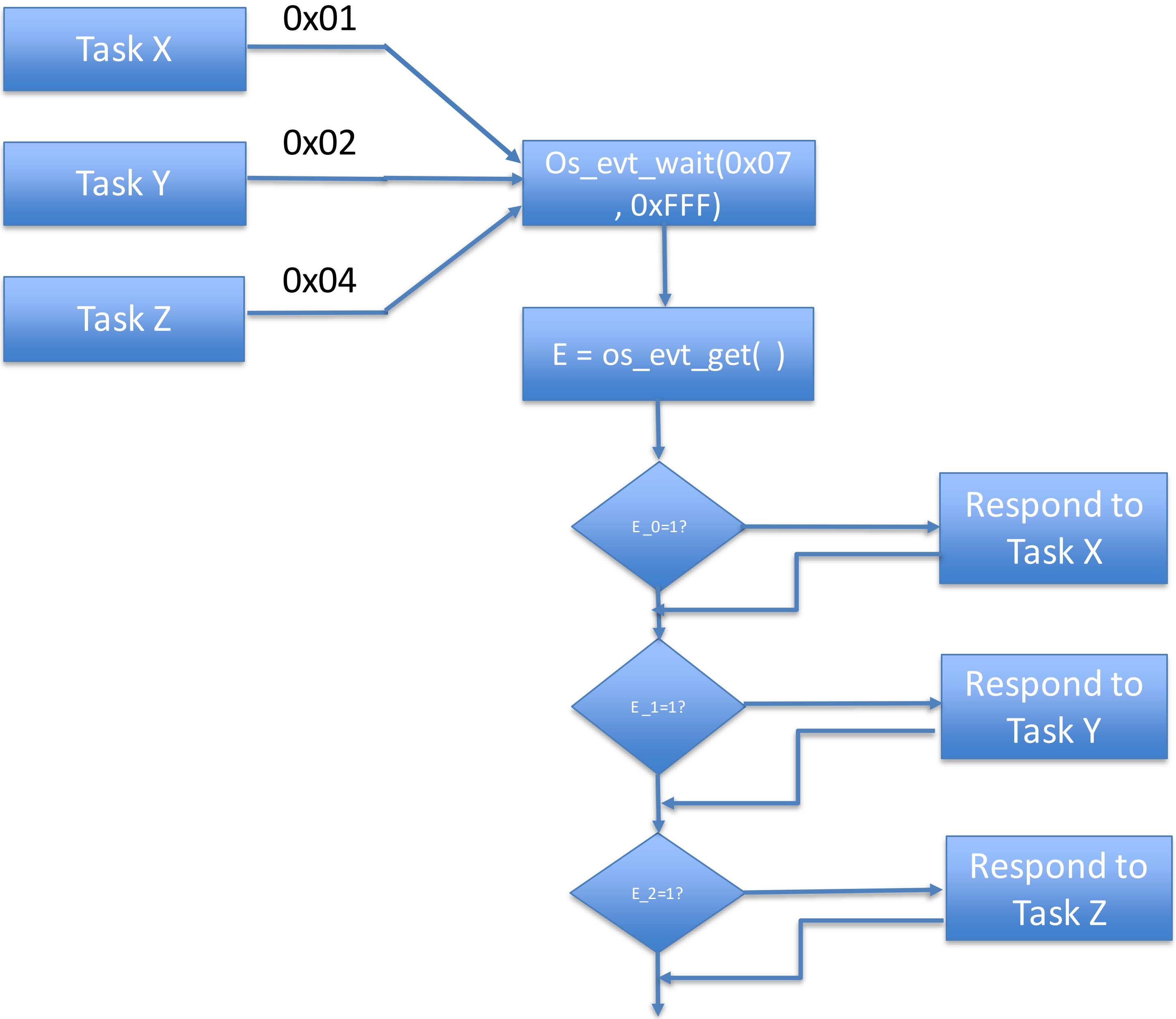


Event communication

Functions for Event Communications

Functions	Description
Os_evg_wait_or	Wait until any of the require flags are received
Os_evt_get	Obtained the pattern value of the event received
Os_evt_set	Send an event pattern to a task
Os_evt_clr	Clear an event from a task
Os_evt_wait_and	Wait until all the required flags are received

Event communications



Task X

0x01

Task Y

0x02

Task Z

0x04

Os_evt_wait(0x07 , 0xFFF)

E = os_evt_get( )

E _0=1?

Respond to Task X

E _1=1?

Respond to Task Y

E_2=1?

Respond to Task Z

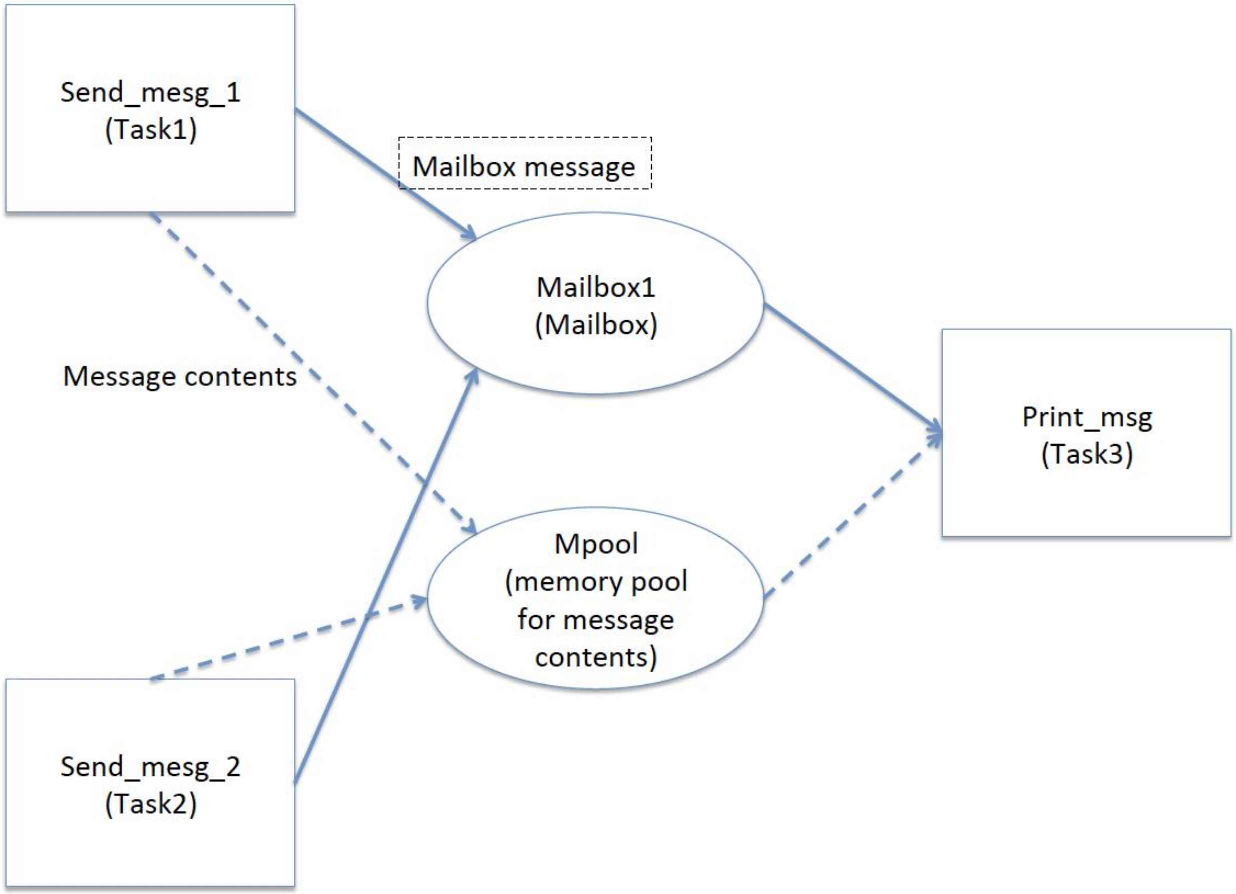
Mailboxes

Mailboxes

The mailboxes is a simple mechanism for asynchronous communication. Some architectures define mailbox registers. These mailboxes registers have a fixed number of bits and can be used for small messages.

Simple version: one that holds only one message at time. Two items: the message itself and a mail ready flag. The flag is true when a message has been put into the mailbox and cleared when the message is removed.

A simple demonstration of using the mailbox feature to transfer messages



Functions for Mailbox and Messaging Operation

Mailbox Functions	Description
Os_mbx_declare	Create a macro to define a mailbox object
Os_mbx_init	Initialize a mailbox object
Os_mbx_send	Sean a message pointer to a mailbox object
Os_mbx_wait	Wait for a message from a mailbox object. If a message is available, get the pointer of the message
Os_mbx_check	Check how many message can still be added to a mailbox object
_declare_box	Declare a memory pool for fixed block size allocation
_init_box	Initialize a fixed block size memory pool
_alloc_box	Allocate a block of memory from the memory pool
_free_box	Return the allocated memory block to the memory pool