



# The Real-Time Operating System $\mu$ COS-II

Enric Pastor

Dept. Arquitectura de Computadors

## $\mu$ C/OS-II Overview

- $\mu$ C/OS-II
  - Task Management
  - Rate Monotonic Scheduling
  - Memory Management
- $\mu$ C/GUI
- $\mu$ C/FS

## Books and Resources

- MicroC/OS-II The Real-Time Kernel Second Edition
  - Jean J. Labrosse
  - Copy available in Library
- MicroC/OS-II e-book (1<sup>st</sup> edition)
  - Jean J. Labrosse
  - Will be on the course website shortly



**μC/OS-II**

## μC/OS - II

- μC/OS-II is a highly portable, ROMable, very scalable, preemptive real-time, deterministic, multitasking kernel
- It can manage up to 64 tasks (56 user tasks available)
- It has connectivity with μC/GUI and μC/FS (GUI and File Systems for μC/OS-II)
- It is ported to more than 100 microprocessors and microcontrollers
- It is simple to use and simple to implement but very effective compared to the price/performance ratio.
- It supports all type of processors from 8-bit to 64-bit



## Task Management – Services

- Task Feature
- Task Creation
- Task Stack & Stack Checking
- Task Deletion
- Change a Task's Priority
- Suspend and Resume a Task
- Get Information about a Task



## Task Feature

- $\mu\text{C}/\text{OS-II}$  can manage up to 64 tasks.
- The **four highest priority tasks** and the **four lowest priority tasks** are reserved for its own use. This leaves us with **56 application tasks**.
- The lower the value of the priority, the higher the priority of the task. (Something on the lines of **Rate Monotonic Scheduling**)
- The task priority number also serves as the task identifier



## Rate Monotonic Scheduling

- In Rate Monotonic Scheduling tasks with the highest rate of execution are given the highest priority
- Assumptions:
  - All tasks are periodic
  - Tasks do not synchronize with one another, share resources, etc.
  - Preemptive scheduling is used (always runs the highest priority task that is ready)
- Under these assumptions, let  $n$  be the number of tasks,  $E_i$  be the execution time of task  $i$ , and  $T_i$  be the period of task  $i$ . Then, all deadlines will be met if the following inequality is satisfied:

$$\sum E_i / T_i \leq n(2^{1/n} - 1)$$

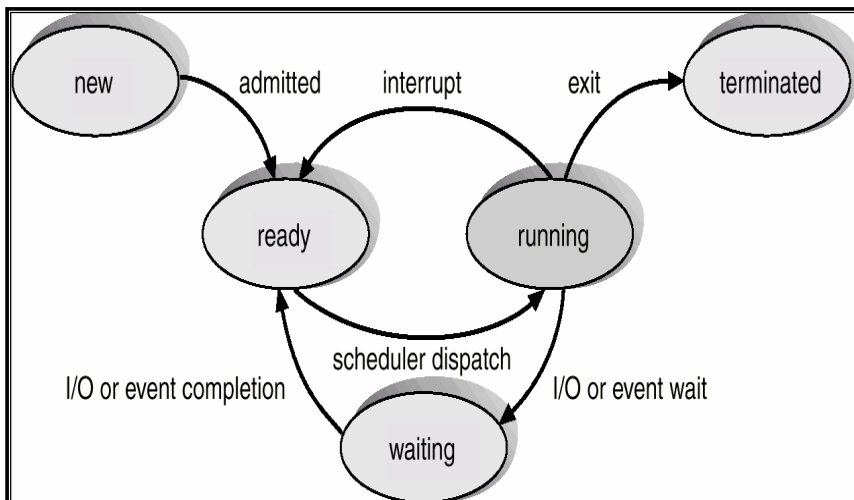


## Rate Monotonic Scheduling: Example

- Suppose we have 3 tasks. Task 1 runs at 100 Hz and takes 2 ms. Task 2 runs at 50 Hz and takes 1 ms. Task 3 runs at 66.7 Hz and takes 7 ms. Apply RMS theory...  
 $(2/10) + (1/20) + (7/15) = 0.717 \leq 3(2^{1/3} - 1) = 0.780$ 
  - Thus, all the deadlines will be met
- General Solution?
  - As  $n \rightarrow \infty$ , the right-hand side of the inequality goes to  $\ln(2) = 0.6931$ . Thus, you should design your system to use less than 60-70% of the CPU



## Process Cycle

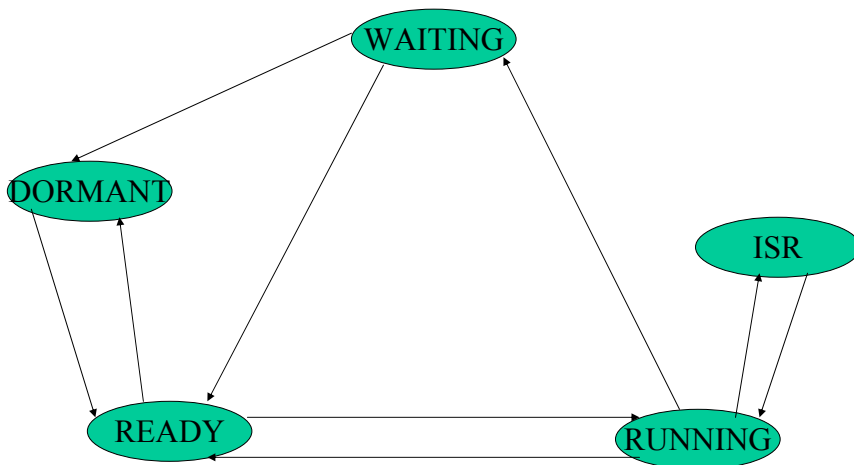


## Task Creation

- Two functions for creating a task:
  - *OSTaskCreate()*
  - *OSTaskCreateExt()*



## Task Management



## Task Management

- After the task is created, the task has to get a stack in which it will store its data
- A stack must consist of contiguous memory locations
- It is necessary to determine how much stack space a task actually uses.
- Deleting a task means the task will be returned to its **dormant** state and does not mean that the code for the task will be deleted. The calling task can delete itself.
- If another task tries to delete the current task, the resources are not freed and thus are lost. So the task has to delete itself after it uses its resources



## Task Management (contd..)

- Priority of the calling task or another task can be changed at run time.
- A task can suspend itself or another task, a suspended task can resume itself
- A task can obtain information about itself or other tasks. This information can be used to know what the task is doing at a particular time.



## Memory Management

- The Memory management includes:
  - Initializing the Memory Manager
  - Creating a Memory Partition
  - Obtaining Status of a Memory Partition
  - Obtaining a Memory Block
  - Returning a Memory Block
  - Waiting for Memory Blocks from a Memory Partition



## Memory Management

- Each **memory partition** consists of several fixed-sized **memory blocks**
- A task obtains memory blocks from the memory partition
  - A task must create a memory partition before it can be used
- Allocation and de-allocation of these fixed-sized memory blocks is done in constant time and is deterministic
- Multiple memory partitions can exist, so a task can obtain memory blocks of different sizes
- A specific memory block should be returned to its memory partition from which it came



## Time Management

- **Clock Tick:** A clock tick is a periodic time source to keep track of time delays and time outs.
  - Tick intervals: 10 ~ 100 ms.
  - The faster the tick rate, the higher the overhead imposed on the system.
- When ever a clock tick occurs  $\mu\text{C}/\text{OS-II}$  increments a 32-bit counter
  - The counter starts at zero, and rolls over to 4,294,967,295 ( $2^{32}-1$ ) ticks.
- A task can be delayed and a delayed task can also be resumed



## Time Management

- Five services:
  - OSTimeDLY()
  - OSTimeDLYHMSM()
  - OSTimeDlyResume()
  - OSTimeGet()
  - OSTimeSet()



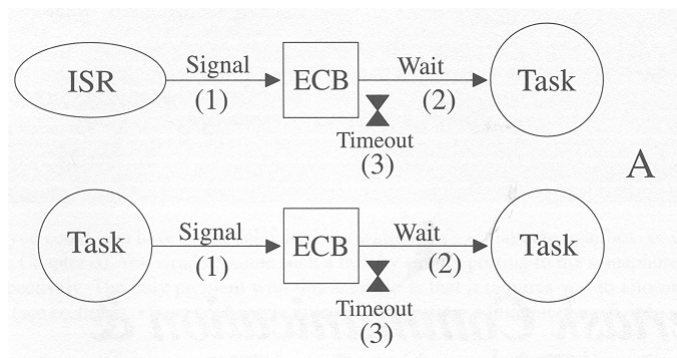
## Inter-task communication

- Inter-task or inter process communication in  $\mu\text{C}/\text{OS}$  takes place using
  - Semaphores
  - Message mailbox
  - Message queues
- Tasks and Interrupt service routines (ISR) can interact with each other through an **ECB** (event control block)



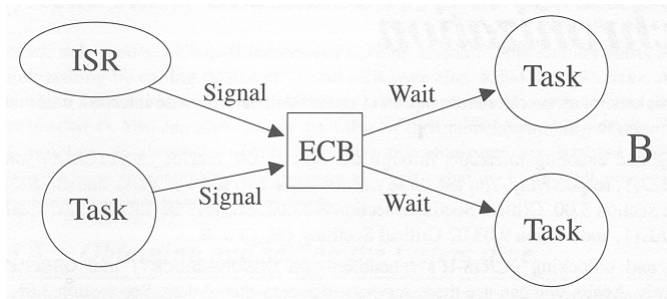
## Inter-task communication

- Single task waiting



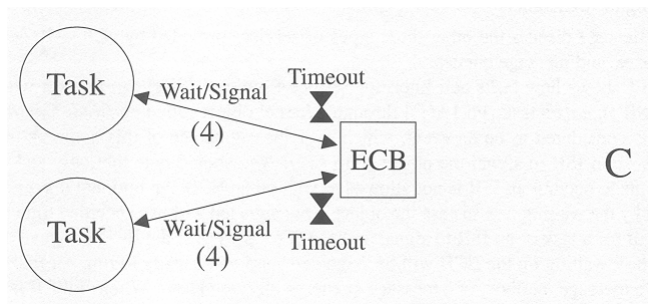
## Inter-task communication

- Multiple tasks waiting and signaling



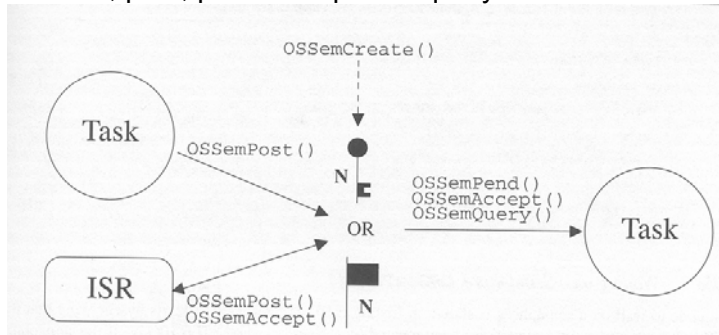
## Inter-task communication

- Tasks can wait and signal along with an optional time out



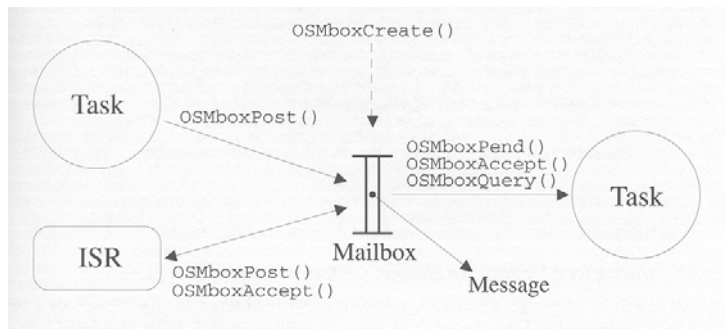
## Inter-task communication

- $\mu$ C/OS-II semaphores consist of two elements
  - 16-bit unsigned integer count
  - list of tasks waiting for semaphore
- $\mu$ C/OSII provides
  - Create, post, pend accept and query services



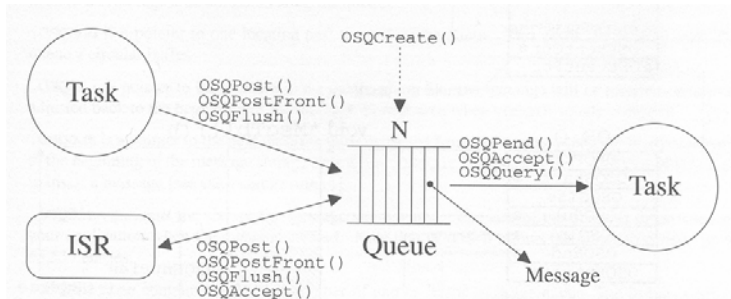
## Inter-task communication

- $\mu$ C/OS-II message-mailboxes: an  $\mu$ C/OSII object that allows a task or ISR to send a pointer sized variable (pointing to a message) to another task.



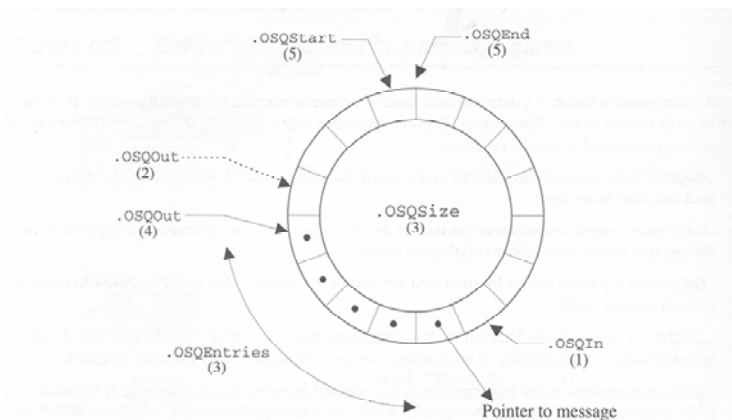
## Inter-task communication

- $\mu\text{C}/\text{OS-II}$  message-queues
- Available services: Create, Post (FIFO), PostFront (LIFO), Pend, Accept, Query, Flush
  - $N = \#$ of entries in the queue: Queue full if Post or PostFront called  $N$  times before a Pend or Accept



## Inter-task communication

- $\mu\text{C}/\text{OS-II}$  message-queues organized as circular buffers.



## Writing Applications Under $\mu$ C/OS-II

- Tasks running under a multitasking kernel should be written in one of two ways:

- A non-returning forever loop. For example:

```
void Task (void *)
{
    DoInitStuff();
    while (1)
    {
        do this;
        do that;
        do the other thing;
        call OS service (); // e.g. OSTimeDelay, OSSemPend, etc.
    }
}
```



## Writing Applications Under $\mu$ C/OS-II

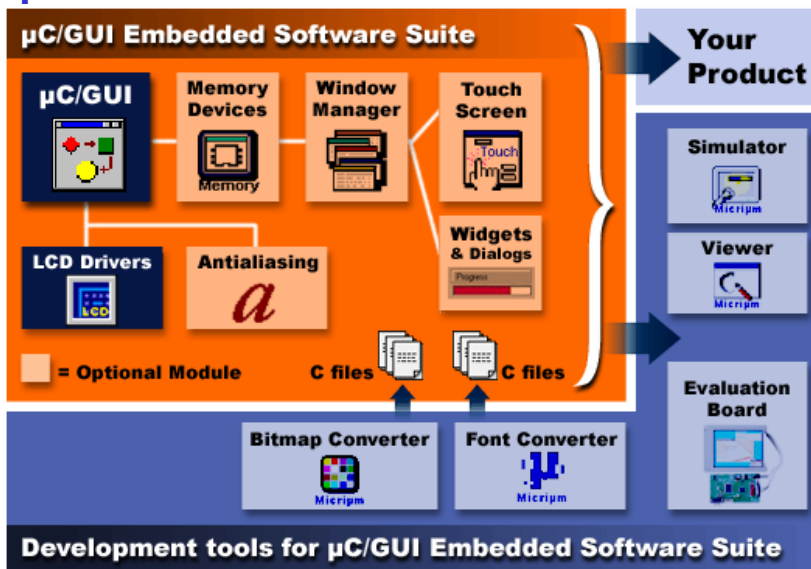
- A task that deletes itself after running. For example:

```
void Task (void *)
{
    do this;
    do that;
    do the other thing;
    call OS service (); // e.g. OSTimeDelay, OSSemPend, etc.
    OSTaskDelete(); // Ask the OS to delete the task
}
```



# μC/GUI

## μC/GUI



## μCGUI (Contd..)

- **Memory Devices:** Frame buffer, the image of the windows etc are drawn on this buffer and when the drawing is completed, this is copied to the touch screen. This helps in preventing the flickering of the screen.
- **Window Manager:** It manages the windows that are created in the GUI. It handles all the mouse and keyboard events (touch screen in this case).
- **Touch Screen:** Touch Screen drivers
- LCD Drivers
- **Anti-aliasing:** This smoothens the fonts and other graphical entities. It gives a pleasant affect to the eye instead of the rugged look.



## μCGUI (contd..)

- **Widgets and Dialogs:** A library to create the widgets (buttons, textboxes, etc) and dialog boxes. Reduces lot of effort to build everything using pixels and lines
- **Font Converter:** It converts a general font format (ttf, utf, etc) to the μCGUI compatible fonts
- **Bitmap Converter:** It converts the 32-bit bitmap images to the compatible bitmap image that is used on μCGUI
- μCGUI supports all the processors from 8-bit to 16-bit



## μC/FS

## μC/FS

- μC/FS is a FAT file system which can be used on any media
- Basic hardware access functions has to be provided
- MS-DOS/MS-Windows compatible FAT12 and FAT16 support
- Multiple device driver support allows to access different types of hardware with the file system at the same time
- Multiple media support. A device driver allows you to access different medias at the same time
- OS support: μC/FS can easily be integrated into any OS

## μC/FS Device Drivers

- μC/FS has been designed to cooperate with any kind of hardware. To use a specific hardware with μC/FS, a device driver for that hardware is required.
- The device driver consists of basic I/O functions for accessing the hardware and a global table, which holds pointers to these functions. Available drivers are:
  - RAM disk
  - SMC (Smart Card)
  - MMC (Multimedia Card)
  - SD (Secure Digital)
  - CF (Compact Flash)
  - Windows (used by the Visual C++ demonstration and simulation code)
  - IDE Hard Disk

