Real Time Operating systems (RTOS) concepts

Abu Bakr Mohamed Ramadan
Eng.abubakr88@gmail.com

Content:

- **Mutual Exclusion .**
- **Disabling and Enabling the interrupts.**
- **Disabling and Enabling the Scheduling.**
- **Semaphores.**
- **Semaphore parameters, and data structures.**
- **Semaphore Types.**
- **References and Read more**

Mutual Exclusion

- Shared Data is important for tasks to communicate.
- Mutual Exclusion access is a must when using any shared resource.
- Examples on Mutual Exclusion methods are:
  - Disable and enable interrupts,
  - Disabling Scheduling, and
  - Using Semaphores.

Disabling and Enabling the interrupts.

- Most of Systems have this technique.
- Example; µC/OS-II uses to micros : ㄴㅈ

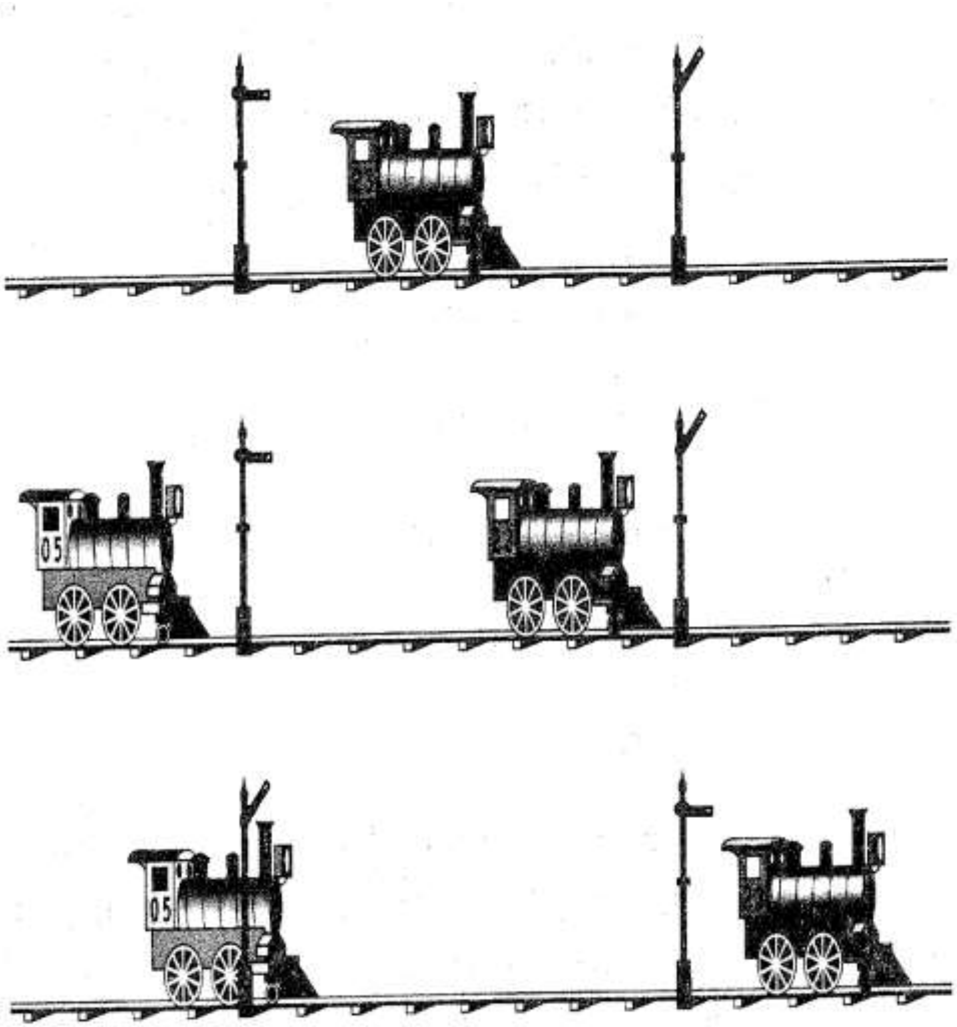
  -
    ```
    OS_ENTER_CRITICAL();                    //Disable interrupts,
    /*
            Read/Write to the shared resource,
    */
    OS_EXIT_CRITICAL();                     //Reenable interrupts.
    ```
- Disabling the interrupt for long time affect the response to your system which known by **Interrupt Latency**.
- **Interrupt Latency:** is The time taken by a system to respond to an interrupt.
- So disable the interrupt should be for as little time as possible.
- This is the only way for a task to share a variable with ISR.

Disabling and Enabling the Scheduling.

- If we don’t share data with any ISR, then it’s better to disable and enable scheduling.
- While the scheduler is locked, the interrupts is enabled, and if interrupt happen, the ISR is executed immediately.
- As the scheduling is disabled, when the ISR finish, the kernel will return to the interrupted task not the highest priority one.
- Example; µC/OS-II uses to micros :ㄴㅈ

  -
    ```
    OSSchedLock();                              //Disable Scheduling,
      /*
        Read/Write to the shared resource,
      */
    OSSchedunLock();                            //Reenable Scheduling.
    ```
- Disabling the scheduler also is not the best solution,

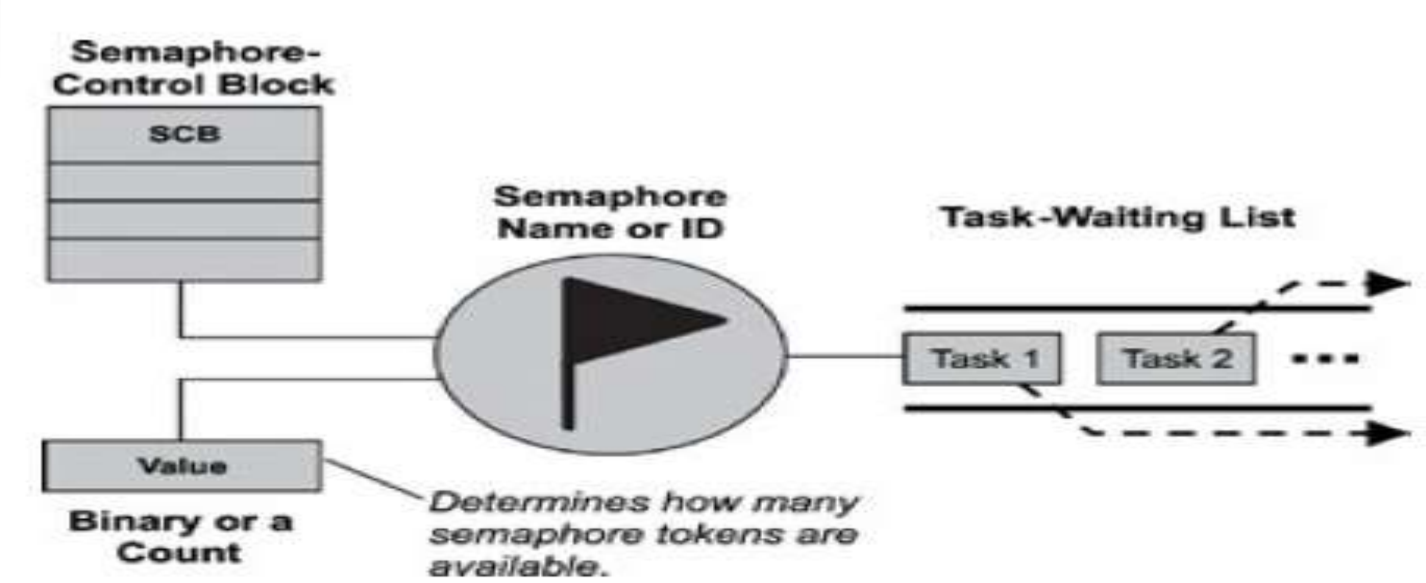
Semaphores



Semaphores

- Is a kernel object that one or more threads of execution can acquire or release for the purposes of synchronization or mutual exclusion.
- A semaphore is like a key that enables a task to carry out some operation or to access a resource.
- When a task acquires the semaphore,
  - No other task can access the resource that is protected by the semaphore.
  - Other tasks acquire the semaphore will be suspended until the semaphore is released by it’s current owner.

Semaphore parameters and data structures.



- When a semaphore is created, the kernel assigns to it:
  - an associated semaphore control block (SCB),
  - a unique ID,
  - a value (binary or a count) depending on it’s type,
  - a task-waiting list,

Semaphore Types

- **Binary Semaphore:**
  - It’s value = 0, if it’s not available.
  - It’s value = 1, if it’s available.
- **Counting Semaphore.**
  - It’s value = 0, if it’s not available.
  - It’s value > 0, if it’s available.

References and Read more:

- **Real-Time Concepts for Embedded Systems book** by Qing Li and Carolyn.
  - http://www.e-reading.club/book.php?book=102147

- **An Embedded Software Primer** by David E. Simon.
  - http://www.amazon.com/Embedded-Software-Primer-David-Simon/dp/020161569X

- **Linux Kernel Embedded Systems Building Blocks 2e** by Jean J. Labrosse.
  - http://www.amazon.com/Embedded-Systems-Building-Blocks-Ready/dp/0879306041

- **FreeRTOS website.**
  - http://www.freertos.org