 <b>UNIVERSIDAD NACIONAL DE COLOMBIA</b>	<b>GESTIÓN DE LABORATORIOS</b>	Código:
		Versión:
	<b>SISTEMAS EN TIEMPO REAL - PRACTICA No 1</b>	Página 1 de 8

## **GUÍA #1: INTRODUCCIÓN A LOS SISTEMAS EN TIEMPO REAL CON ARDUINO**

### **1. INTRODUCCIÓN**

En esta guía se abordan dos temas importantes de los sistemas operativos en tiempo real. Uno es la programación concurrente, por medio de la programación de tareas por hilos. El otro es la sincronización de tareas por medio del manejo de semáforos. Se aborda además la programación de hilos y semáforos haciendo el uso de la librería FreeRTOS para Arduino.

### **2. OBJETIVOS**

- Analizar los conceptos básicos de sistemas en tiempo real.
- Comprender el funcionamiento de la programación por hilos.
- Identificar la importancia de los semáforos en la programación por hilos.

### **3. ANTECEDENTES**

Para que un sistema sea considerado “de tiempo real”, principalmente, debe cumplir con sus tareas en tiempos específicos, o sea que tiene unas restricciones de respuesta temporales. A diferencia de los sistemas informáticos convencionales, el retardo en la respuesta de un sistema en tiempo real es crítica, ya que estos sistemas se utilizan más que todo en aplicaciones que requieren alta precisión temporal para prevenir riesgos (e.g. procesos que involucren reacciones químicas, procedimientos en hospitales, aplicaciones de vuelo, etc.). Otras características de los sistemas en tiempo real son la programación concurrente, la comunicación y sincronización entre tareas y la tolerancia fallos.

#### **Programación concurrente**

Al desarrollar una aplicación bajo algún lenguaje de programación (C/C++, Java, Python), se sabe que el algoritmo que se escriba se llevará a cabo en el orden en el que se escribió, y si se programa más de una tarea, siempre se ejecutará antes la tarea que se codificó primero. Esto es lo que se conoce como programación secuencial.

Ahora supóngase que se tienen dos tareas que necesitan ser atendidas rápidamente, si se tuvieran dos procesadores, se podría asignar cada tarea a un procesador y así serían atendidas de manera paralela. Pero si se tiene un solo procesador, hay que buscar una manera de que las tareas no tengan que esperar a que el algoritmo de la otra tarea se ejecute para empezar ellas a ejecutarse. Esa herramienta es la **programación concurrente**, que, por medio de un administrador de tiempos (algoritmo de planificación), le da a cada tarea una ventana de tiempo para que se ejecute de manera parcial o total. Se apoya en la **programación por hilos**, que consiste en programar tareas en bucles de ejecución independientes, donde cada tarea pertenece a un “hilo de programación”.

RAFAGA	TIEMPO LLEGADA	REQUERIMIENTOS DE CPU(ms)
R1	0	16
R2	1	3
R3	2	2

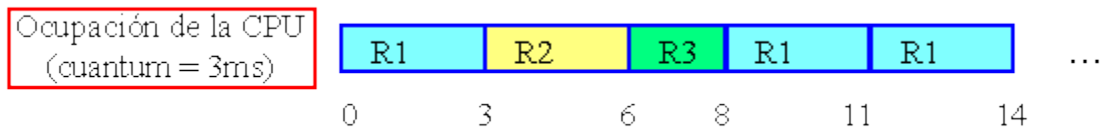


Figura 1 y 2. Algoritmo de programación para programación concurrente. Tomadas de [3].

Las Figuras 1 y 2 ilustran el concepto de programación concurrente. Al procesador llega la orden de ejecución de 3 tareas (R1, R2 y R3), correspondientes a hilos diferentes, y por medio de un algoritmo de planificación (*Round Robin* en este caso), se le asigna a cada tarea una ventana de tiempo de ejecución. La tarea R1, que es más larga que las demás, se pausa luego de acabar su primera ventana de tiempo y queda en espera de su siguiente turno.


### Comunicación y sincronización de tareas

Durante la ejecución de un Sistema Operativo de Tiempo Real (RTOS), el procesador no se va a encontrar siempre con la facilidad de atender solo una tarea a la vez, y muchas veces las tareas deben compartir información entre ellas o dependerán de algún recurso en común para su ejecución, por lo que es necesario que haya métodos de comunicación y sincronización entre las tareas. Estos métodos existen, por un lado, para facilitar el intercambio de información entre las tareas, y por otro lado, para prevenir colapsos, bucles infinitos o corrupción de la información debido a condiciones de carrera. Uno de los métodos de sincronización a trabajar en esta guía es por semáforos, en el cual el acceso al recurso se da por turnos, luego de que una tarea haya dejado de utilizar el recurso, el semáforo da "luz verde" a la siguiente tarea en la lista de espera para que acceda a él.

### Fiabilidad y tolerancia a fallos

Los sistemas en tiempo real deben ser fiables, es decir, deben tener la característica de ajustarse a un comportamiento definido como correcto con base en unas especificaciones dadas. Cuando se trabaja en ambientes que dependen totalmente del funcionamiento constante del sistema en tiempo real, reiniciar las tareas o cancelarlas **nunca** es una opción ya que se compromete la eficiencia de las operaciones o, en algunos casos, vidas humanas.

Por estas razones, debe haber mecanismos de prevención y detección de errores para hacer que el sistema sea tolerante a fallos.

 <b>UNIVERSIDAD NACIONAL DE COLOMBIA</b>	<b>GESTIÓN DE LABORATORIOS</b>	Código:
		Versión:
	<b>SISTEMAS EN TIEMPO REAL - PRACTICA No 1</b>	Página 3 de 8

Para la tolerancia a fallos por software, existen dos métodos altamente utilizados, uno es la programación con N-versiones y otro es la programación para la detección y recuperación. En la programación por N-versiones, se busca tener más de una manera diferente de desarrollar la misma tarea y por medio de un consenso llegar a la respuesta correcta, así, en caso de que una de las maneras de un resultado bastante diferente a los otros, se descarta inmediatamente. La programación para la detección y recuperación busca detectar resultados errados en algún punto del sistema, ya sea comparando resultados esperados o con temporizadores, y luego devolver al sistema a un punto de operación correcto.

#### 4. DESARROLLO

Las actividades de esta guía van principalmente enfocadas al aprendizaje de la programación por hilos en Arduino y el uso de semáforos.

Antes que nada, es necesario instalar la librería “FreeRTOS” para la programación en tiempo real. Para esto, se deben realizar los siguientes pasos:

1. Seleccionar la opción Programa > Incluir Librería > Gestionar Librería.

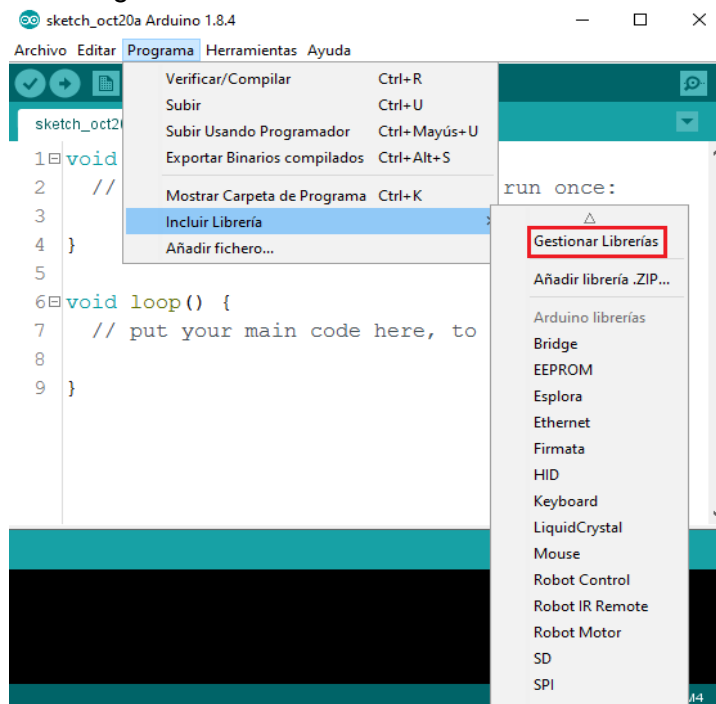


Figura 3. Opción Gestionar Librerías en Arduino IDE.

2. En la ventana que se despliega escribir en la barra de texto “FreeRTOS”, seleccionar la versión más actualizada de la librería y seleccionar la opción Instalar.

 <b>UNIVERSIDAD NACIONAL DE COLOMBIA</b>	<b>GESTIÓN DE LABORATORIOS</b>	Código:
		Versión:
	<b>SISTEMAS EN TIEMPO REAL - PRACTICA No 1</b>	Página 4 de 8

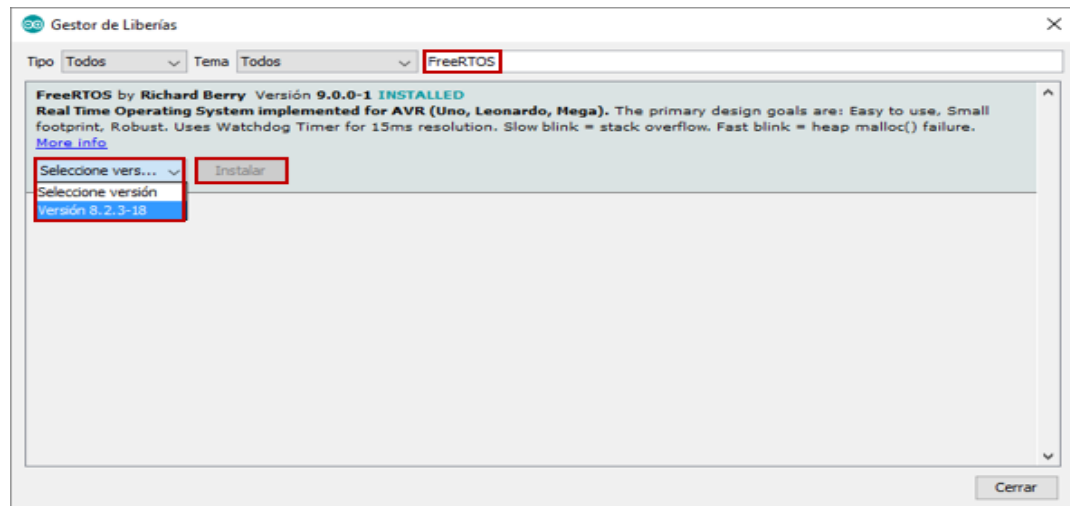


Figura 4. Instalación de la librería FreeRTOS.

Luego de estos pasos, la librería ya estará lista para utilizarse.

#### 4.1. ACTIVIDADES:

La Figura 5, muestra los comandos necesarios para crear un hilo de programación.

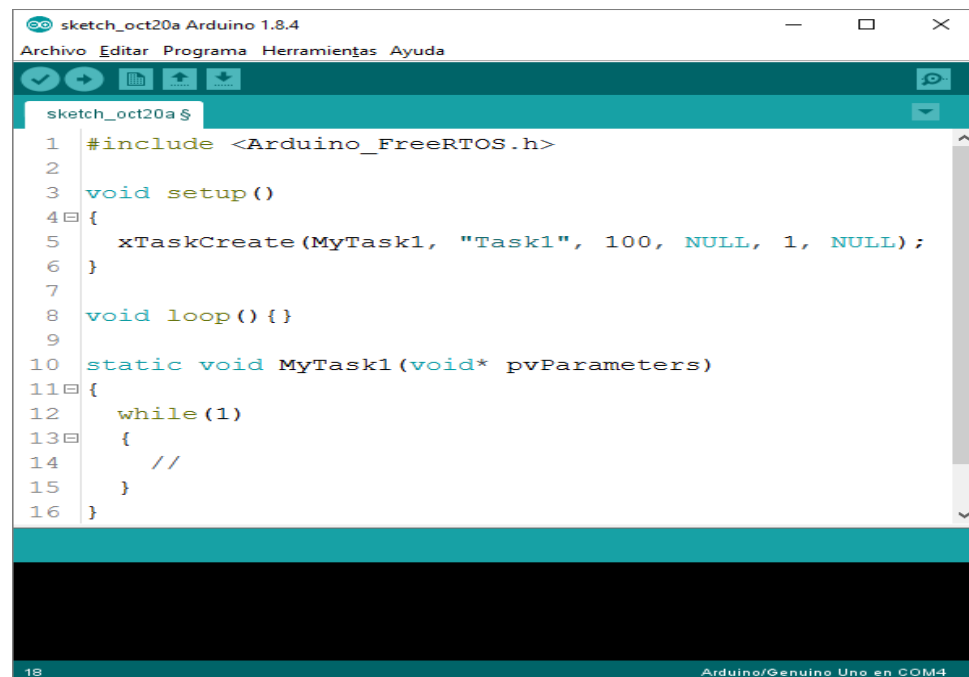


Figura 5. Creación de un hilo de programación en Arduino IDE.

**#include <Arduino\_FreeRTOS.h>:** se encarga de llamar la librería FreeRTOS para ser utilizada en el código.

 <b>UNIVERSIDAD NACIONAL DE COLOMBIA</b>	<b>GESTIÓN DE LABORATORIOS</b>	Código:
		Versión:
	<b>SISTEMAS EN TIEMPO REAL - PRACTICA No 1</b>	Página 5 de 8


- **xTaskCreate(MyTask1, “Task1”, Tamaño de memoria, NULL, Prioridad, NULL):** Con este comando se declara la creación de un hilo de programación. “*MyTask1*” corresponde al nombre de la función que ejecutará el hilo, en la parte inferior de la imagen se muestra la función creada con el mismo nombre. “*Task1*” es simplemente una ayuda visual para el programador, no afecta el desempeño de las tareas dentro de la función. “*Tamaño de memoria*” es un número entero que indica la cantidad de memoria en Kb que se le dará a ese hilo para almacenar información. “*Prioridad*” es un número que permite asignar mayor importancia a un hilo (cuando se tiene más de un hilo) donde 0 es la menor prioridad y configMAX\_PRIORITIES-1 es la mayor prioridad, configMAX\_PRIORITIES es un parámetro interno configurable de la librería (Viene por defecto con configMAX\_PRIORITIES = 4).
- **static void MyTask1(void\* pvParameters){}**: Con este comando se crea la función que se va a ejecutar en el hilo creado. Entre llaves va el código de las tareas que se quieren llevar a cabo dentro de la función y, a diferencia del comando “*void loop()*”, dentro de esta función tiene que crearse un ciclo infinito para que la tarea se repita indefinidamente, sino solo se ejecutará una vez. “*void\* pvParameters*” se refiere a las variables globales del sistema, es decir, a todas las variables que se utilizan y pueden ser compartidas entre hilos, las variables declaradas entre llaves solo existen para esa función.

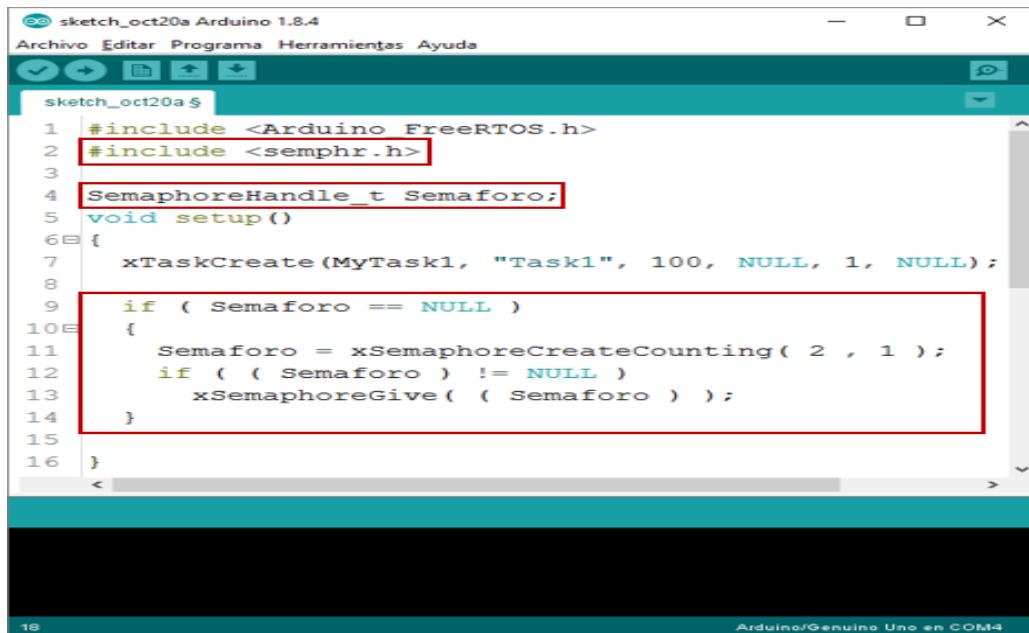
Es importante anotar que, aunque es necesario tener el comando “*void loop()*” en el código, este no se utiliza para nada ya que con el comando “*xTaskCreate()*” se están creando los hilos que ejecutarán las tareas programadas.

Ahora, si se desea utilizar semáforos, se hace por medio de los comandos mostrados en la Figura 6. La Figura muestra los comandos necesarios para crear un semáforo.

**#include <semphr.h>:** Llama todas las funciones correspondientes a los semáforos de la librería FreeRTOS.

- **SemaphoreHandle\_t Semaforo:** Crea una variable de tipo “*SemaphoreHandle\_t*” (proveniente de semphr.h) que guarda todos los parámetros del semáforo con el que se quiere trabajar.
- **xSemaphoreCreateCounting(ConteoMáximo, ConteoInicial):** Llama un tipo de semáforo que funciona por conteo, donde “*ConteoMáximo*” es el valor máximo que puede llegar a tener el contador del semáforo y “*ConteoInicial*” es el valor de inicio del contador. Cada vez que una tarea accede al semáforo, el contador aumenta una unidad, mientras que lo contrario pasa cuando la tarea suelta el semáforo. La cantidad de hilos a los que el semáforo permite utilizar el recurso “al mismo tiempo” se puede calcular con la ecuación ConteoMáximo - ConteoInicial. El condicional en el que está contenido el semáforo en la imagen 6 se utiliza para verificar la existencia de un semáforo previo para crear uno y en caso de ya haber sido creado, enviarlo a su condición inicial.

 <b>UNIVERSIDAD NACIONAL DE COLOMBIA</b>	<b>GESTIÓN DE LABORATORIOS</b>		Código:
			Versión:
	<b>SISTEMAS EN TIEMPO REAL - PRACTICA No 1</b>		Página 6 de 8



```

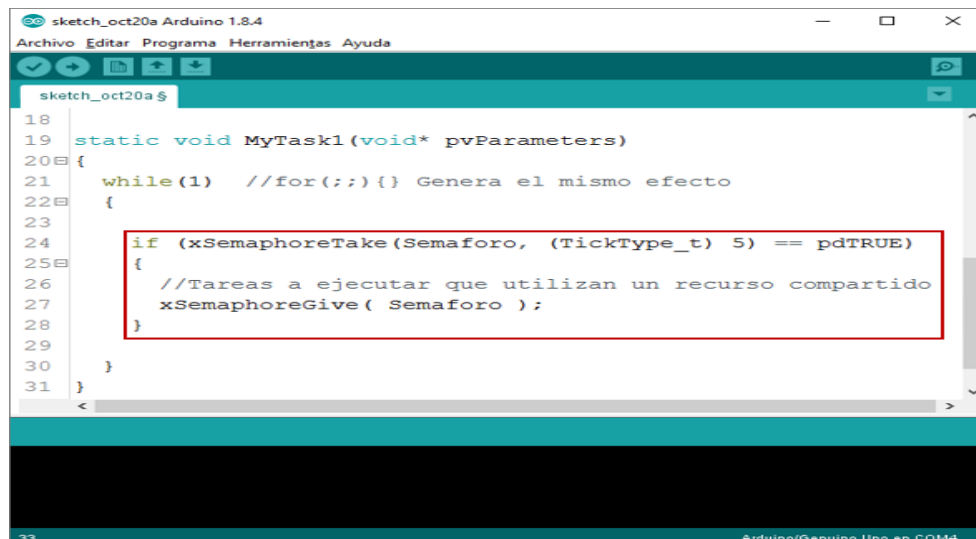
1  #include <Arduino_FreeRTOS.h>
2  #include <semphr.h>
3
4  SemaphoreHandle_t Semaforo;
5  void setup()
6  {
7      xTaskCreate(MyTask1, "Task1", 100, NULL, 1, NULL);
8
9      if ( Semaforo == NULL )
10     {
11         Semaforo = xSemaphoreCreateCounting( 2 , 1 );
12         if ( ( Semaforo ) != NULL )
13             xSemaphoreGive( ( Semaforo ) );
14     }
15 }

```

Figura 6. Creación semáforo en Arduino IDE.

La Figura 7 muestra los comandos básicos para utilizar los semáforos dentro de un hilo:

- **xSemaphoreTake(Semaforo, Timer):** Le permite al hilo que se está ejecutando utilizar el semáforo para “tomar” el recurso compartido. El condicional que maneja este comando se utiliza para esperar cierta cantidad de ciclos de reloj, definida por “*Timer*”, para verificar que el hilo haya tomado el recurso.
- **xSemaphoreGive(Semaforo):** Hace que el hilo “suelte” el recurso compartido y le permita utilizarlo a otro hilo.




```

18
19 static void MyTask1(void* pvParameters)
20 {
21     while(1) //for(;;){} Genera el mismo efecto
22     {
23
24
25         if (xSemaphoreTake(Semaforo, (TickType_t) 5) == pdTRUE)
26         {
27             //Tareas a ejecutar que utilizan un recurso compartido
28             xSemaphoreGive( Semaforo );
29         }
30     }
31 }

```

Figura 7. Uso de semáforo dentro de un hilo en Arduino IDE.

 <b>UNIVERSIDAD NACIONAL DE COLOMBIA</b>	<b>GESTIÓN DE LABORATORIOS</b>	Código:
		Versión:
	<b>SISTEMAS EN TIEMPO REAL - PRACTICA No 1</b>	Página 7 de 8

#### 4.1.1. Actividad: Hilos para bordar tareas

Para esta actividad, deben programar una secuencia de iluminación para 3 LED's. Primero deben realizarlo por medio de programación secuencial (sin la librería) y luego deben realizarlo utilizando hilos (un hilo para cada LED).

Uno de los 3 LED debe parpadear con un período seleccionado por usted, el segundo LED debe parpadear la mitad de este período y el último led debe parpadear a un cuarto del período seleccionado. Todos los LED deben empezar su secuencia "al mismo tiempo". El ciclo de dureza debe ser del 50% (mitad del período encendido y mitad del período apagado).

Al finalizar la programación de la secuencia de las dos maneras, responda las siguientes preguntas:

¿Con cuál método se le hizo más fácil programar las secuencias? ¿Por qué?

---

¿El cambio en la prioridad de los hilos afecta en algo a las tareas programadas?

---

#### 4.1.2. Actividad: Usted si, usted no

En esta actividad, deben utilizar como recurso compartido el puerto serial del Arduino. Deben desarrollar 3 (o más) hilos que utilicen el puerto serial para mostrar mensajes diferentes.

La configuración inicial del puerto serial se realiza en el "*void setup()*" de manera convencional.

Responder las siguientes preguntas:

¿Qué pasa con los mensajes cuando no se utilizan semáforos?

---

¿Qué pasa con los mensajes si el semáforo les permite utilizar el recurso "al mismo tiempo"?

---

¿Cómo afecta la prioridad a la ejecución de los hilos?

---

 <b>UNIVERSIDAD NACIONAL DE COLOMBIA</b>	<b>GESTIÓN DE LABORATORIOS</b>	Código:
		Versión:
	<b>SISTEMAS EN TIEMPO REAL - PRACTICA No 1</b>	Página 8 de 8

Cambie el comando `xSemaphoreCreateCounting(2,1)` por el comando `xSemaphoreCreateMutex()`.

¿Hay alguna diferencia en la ejecución de las tareas? ¿Por qué?

## 4.2. RETO:

Con los conocimientos adquiridos en las actividades anteriores, deberá mostrar por el puerto serial la información leída de 3 (o más) sensores analógicos.

Debe procesar la señal de tal manera que la información que se muestre sea correspondiente al sensor que se utilice (ej. Si se lee un sensor de temperatura, mostrar valores en grados centígrados o Fahrenheit).

Elija cada cuánto se va a mostrar la información de uno de los sensores, y con base en ese tiempo, utilice la mitad de ese tiempo para mostrar la información de otro de los sensores y un cuarto de ese tiempo para mostrar la información del tercer sensor.

## 5. REFERENCIAS

- [1] Universitat De València. Programación de los Sistemas de Tiempo Real. Disponible en internet: [https://www.uv.es/gomis/Apuntes\\_SITR/Programacion.pdf](https://www.uv.es/gomis/Apuntes_SITR/Programacion.pdf)
- [2] Universidad De Córdoba. SISTEMAS EN TIEMPO REAL. Disponible en internet: <http://www.uco.es/~el1orlom/docs/STRtema1.pdf>
- [3] LSI Vitoria – Gasteiz UPV/EHAU. Algoritmos de planificación. Disponible en internet: <https://lsi.vc.ehu.eus/pablogn/docencia/manuales/SO/TemasSOuJaen/PLANIFICACIONDEPROCOSOS/6AlgoritmosdePlanificacionI.htm>
- [4] Referencia (Título de la página. Título del artículo. Disponible en internet:)