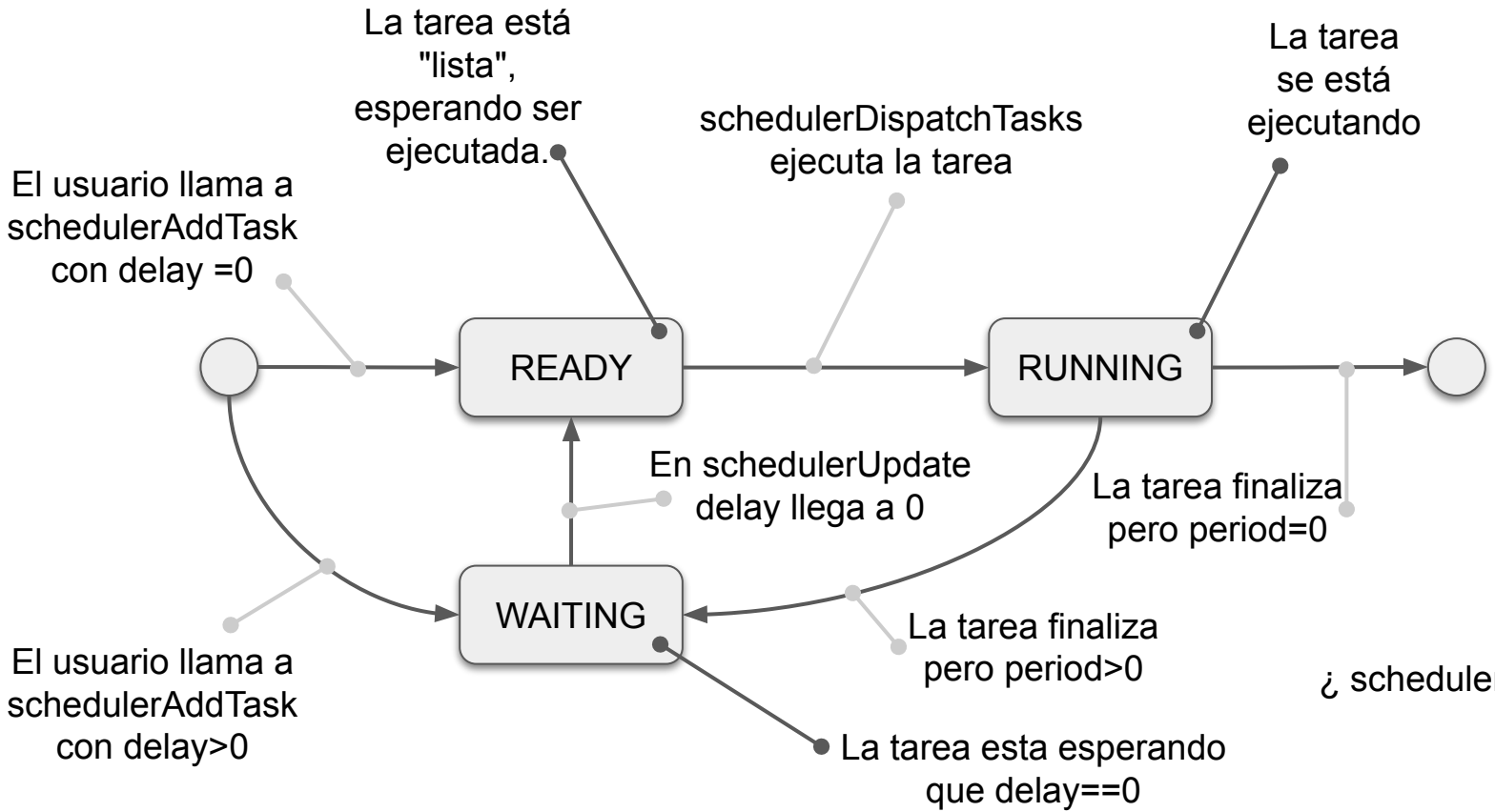
Carrera de Especialización en Sistemas Embebidos

Sistemas Operativos en Tiempo Real

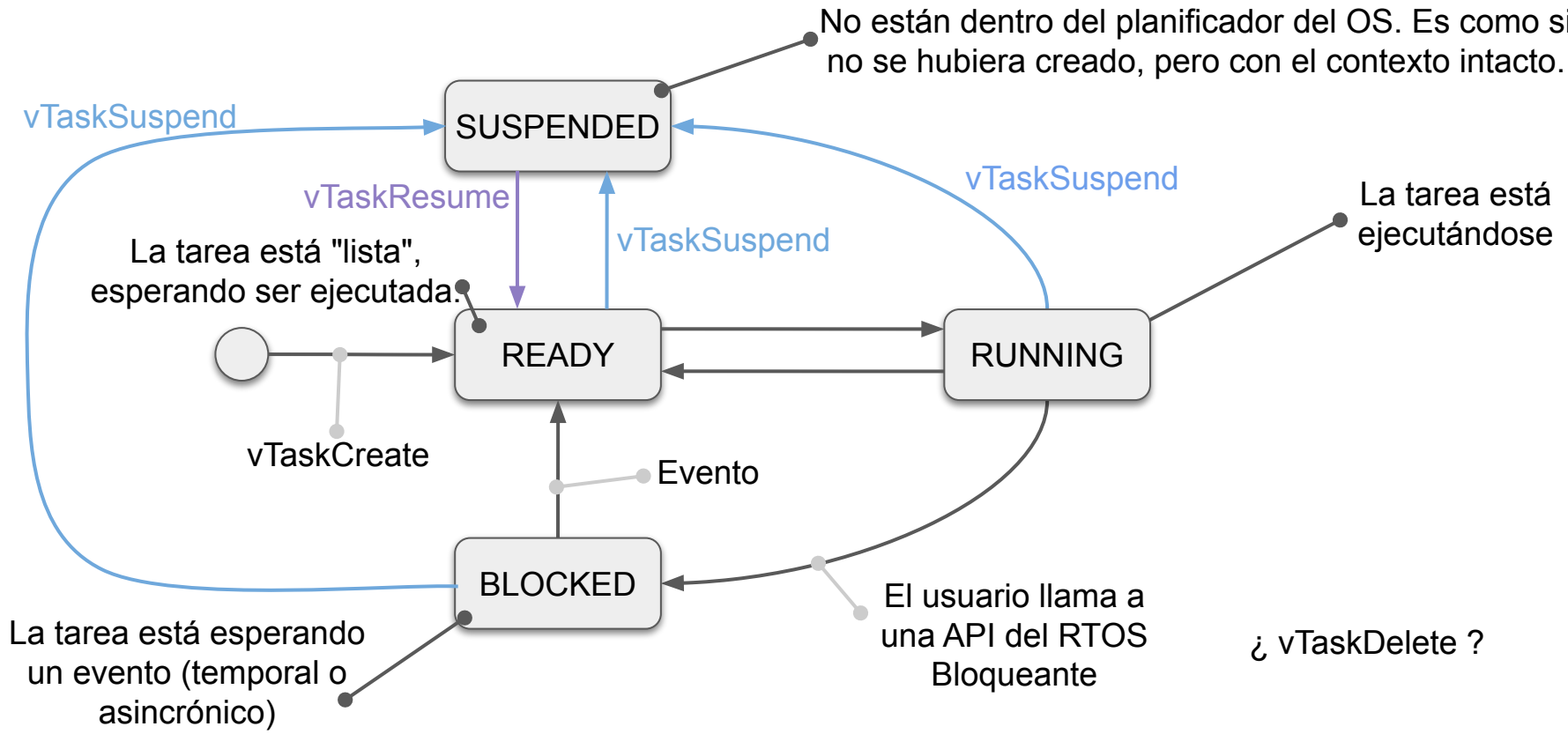
Clase 2: Planificación de tareas

ACSE
Asociación Civil para la Investigación, Promoción y Desarrollo de los Sistemas Electrónicos Embebidos
FACULTAD DE INGENIERIA
Universidad de Buenos Aires
CIM
CADIEEL

SEOS Pont: Estados de tareas equivalente



FreeRTOS: Estados de tareas

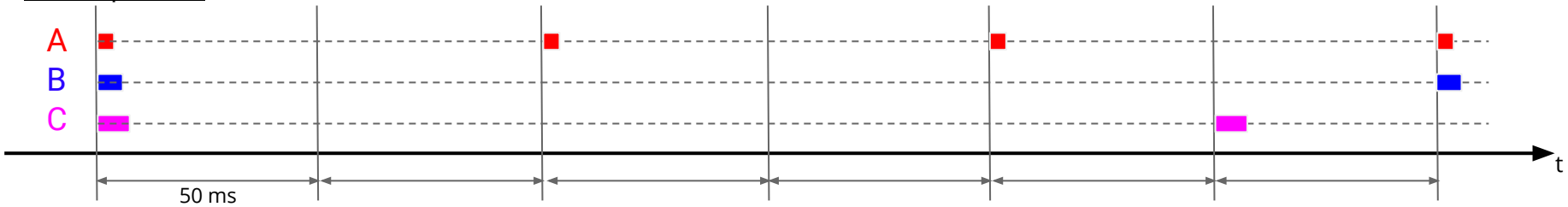


Planificación de tareas: Baremetal con PONT

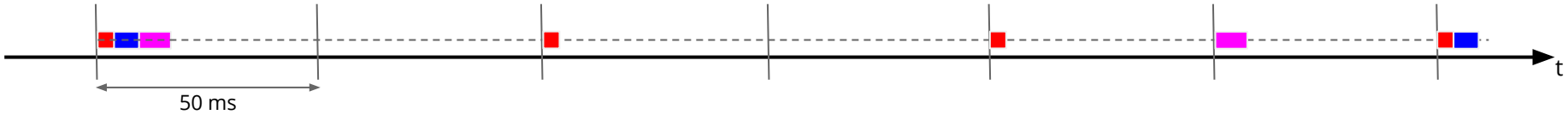
Tarea	Tiempo Computo (Peor caso)	Periodicidad
A	2 ms	100 ms
B	3 ms	300 ms
C	4 ms	250 ms

Base de tiempo= 50 ms

Uno esperaría:



Pero.... sabemos que no ocurre eso:



Depende del orden en que se llame a `schedulerAddTask`

La tarea C posee Jitter en la periodicidad

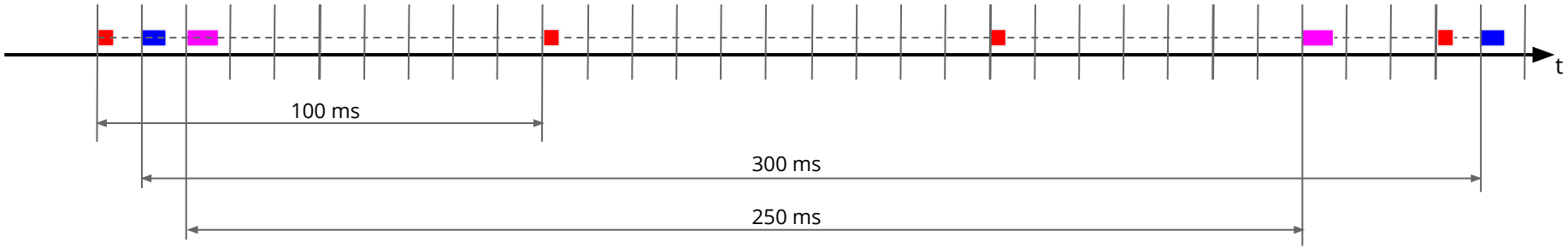
Planificación de tareas: Baremetal con PONT

Tarea	Tiempo Computo (Peor caso)	Periodicidad	Retardo
A	2 ms	100 ms	0 ms
B	3 ms	300 ms	10 ms
C	4 ms	250 ms	20 ms

Base de tiempo= 10 ms

~~Base de tiempo= 50 ms~~

¿ Cómo implementamos un sistema en tiempo real ?

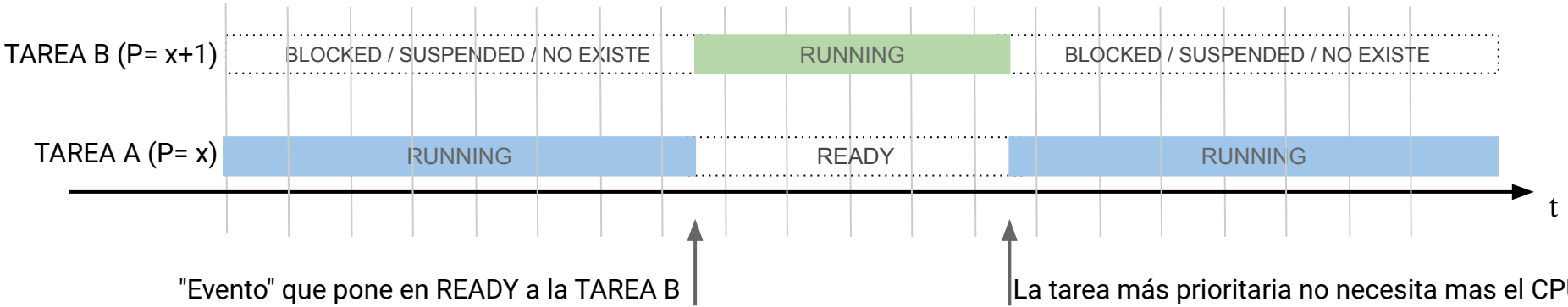


¡¡ Un sistema en Hard Real time es mucho más difícil de diseñar !!

¿ Qué ocurre con las interrupciones ?

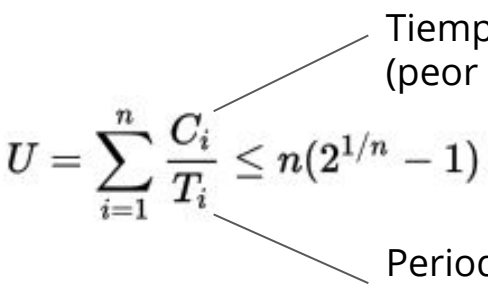
Fixed priority preemptive scheduling

- Fixed priority significa que el kernel NO MODIFICA las prioridades de las tareas
- Preemptive: Significa que es apropiativo
- Este algoritmo permite que, de un conjunto de tareas listas para ser ejecutadas (READY) SIEMPRE ejecute la tarea de mayor prioridad.



Fixed priority preemptive scheduling

- ¿ Cómo se asignan las prioridades ?
- Algoritmo "Rate Monotonic Scheduling" (**RMS**)
  - Se asignan prioridades más altas a tareas más frecuentes.
- Utilización del CPU (modelo válido solo para tareas periódicas)



0 <= U <= B(n) : planificable

B(n) <= U <=1 : no se puede concluir nada

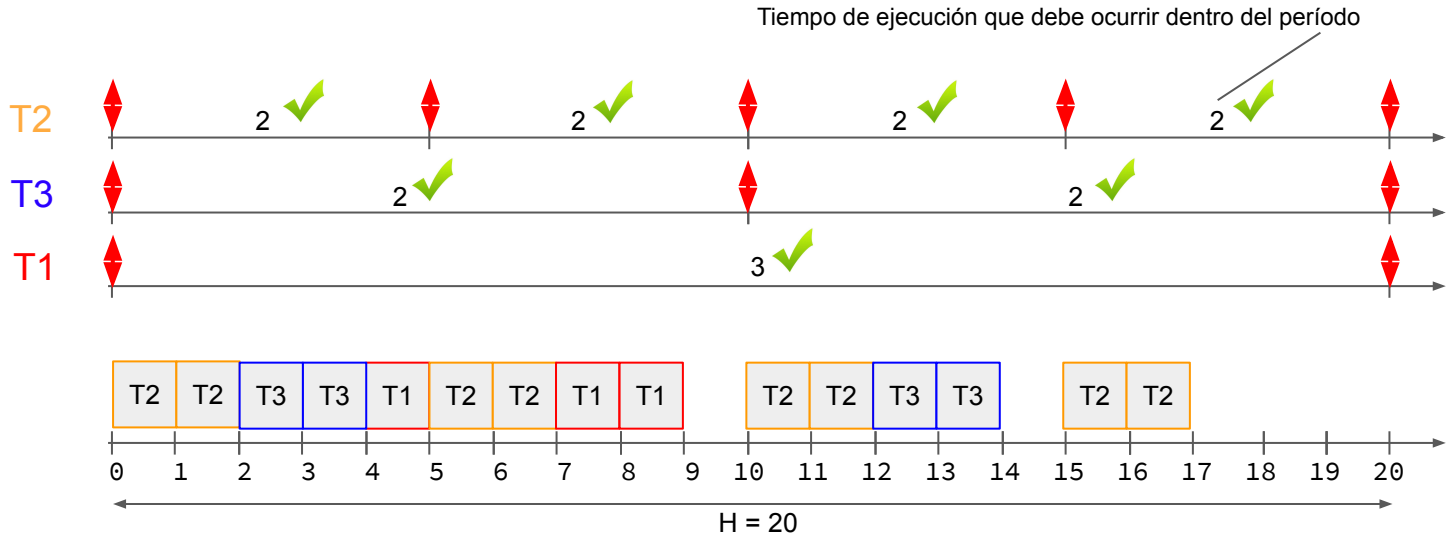
U > 1 : sobrecarga

Fixed priority preemptive scheduling

Tarea	Tiempo Computo (Ci)	Periodicidad (Ti)
T1	3 ms	20 ms
T2	2 ms	5 ms
T3	2 ms	10 ms

Hiper periodo : H = MCM(20, 5, 10)

Usando RMS ordenamos las prioridades: P(T2) > P(T3) > P(T1)



$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$ 3/20 + 2/5 + 2/10 = 15/20 = 75% < 77.9%

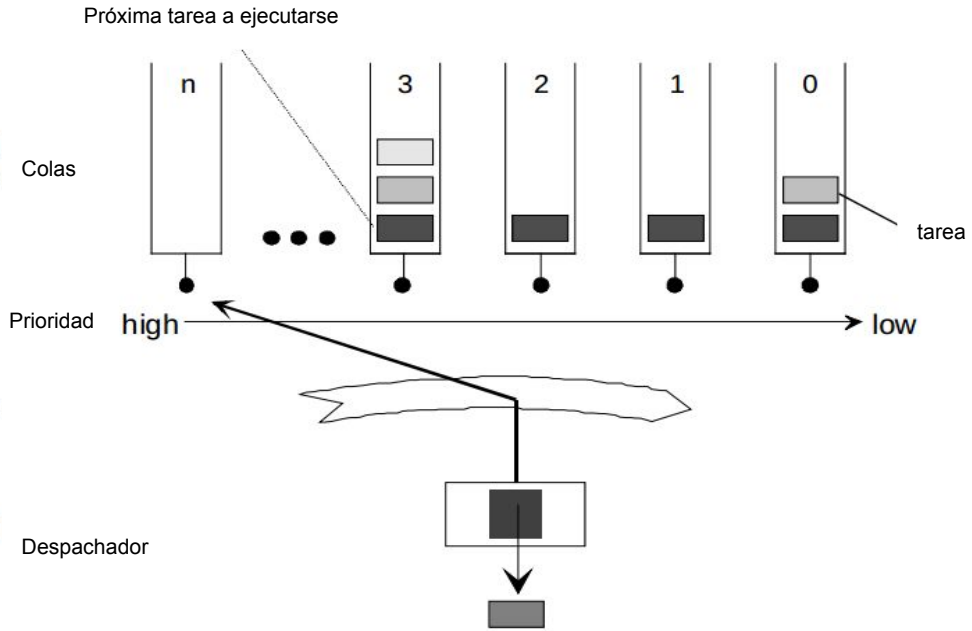
Fixed priority preemptive scheduling

- Algoritmo "Deadline Monotonic Scheduling" (DMS)
  - Se asignan prioridades más altas a tareas deadline relativos más cortos

$$U = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1)$$

- En este caso, en vez de la periodicidad, se utiliza el deadline para el cálculo de la utilización, de igual manera que para realizar un análisis similar al anterior.

FPPS implementación



Prioridad dinámica

- Dynamic priority preemptive scheduling
  - ¿ Cómo se asignan las prioridades ? **No se asignan**.
    Se definen deadlines relativos y el scheduler cambia las prioridades basado en esos valores.

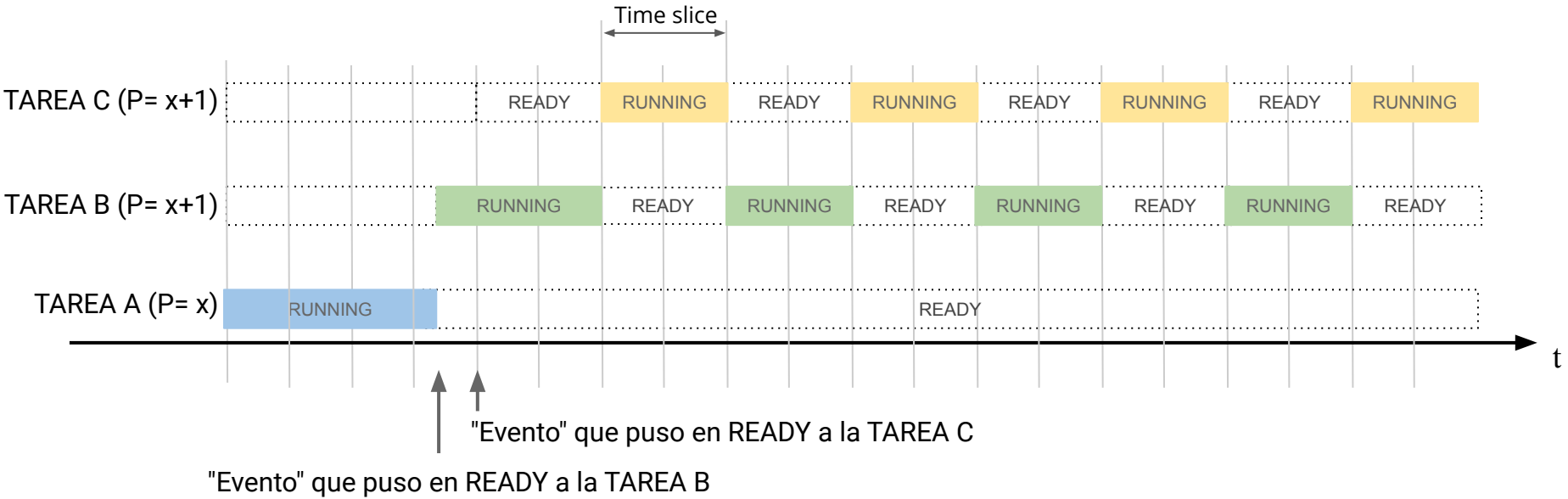
- Algoritmo "Earliest Deadline First" (EDF)
  - El planificador le da mayor prioridad a la tarea que se encuentra más cerca del deadline.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1,$$

- Shortest Job First
- Shortest Remaining Time First
- Longest Remaining Time First

Round Robin

- Para tareas de la misma prioridad, listas para ser ejecutadas, el planificador divide el tiempo de CPU y lo reparte entre ellas.
- Esta opción puede desactivarse.



API: **Arranque del OS**

- `void vTaskStartScheduler( void );`
  - Arranca el planificador del OS.
  - Crea la tarea IDLE.
  - Corre por primera vez el planificador, para evaluar cual tarea hay que ejecutar.

```c
int main(void)
{
	prvSetupHardware();
	xTaskCreate(tareaA, (signed char *) "descripcion",
		configMINIMAL_STACK_SIZE, NULL,
		(tskIDLE_PRIORITY + 1UL),
		(xTaskHandle *) NULL);

	vTaskStartScheduler();

	return 1;
}
```

```c
void tareaA(void *pvParameters)
{
	/* código de la tarea */
}
```

API: **Creación de tareas**

- BaseType_t **xTaskCreate**(...):
  - Crea e inicializa los espacios de memoria para la ejecución de una tarea. La tarea inicia en estado READY.
- Se la puede llamar en cualquier momento.
- Parámetros:
  - pvTaskCode: referencia a la función de C que describe el comportamiento.
  - pcName: Nombre descriptivo.
  - usStackDepth: Tamaño del stack en words. (valor mínimo configMINIMAL_STACK_SIZE)
  - pvParameters: Parámetro opcional a la tarea (permite varias instancias de la misma tarea, con comportamientos distintos)
  - uxPriority : prioridad de la tarea. Debe ser mayor a tskIDLE_PRIORITY
  - pxCreatedTask: Handle de la tarea, si es requerido.

API: **Eliminación de tareas**

- `void` **`vTaskDelete`**`( TaskHandle_t xTask ):`
  - Quita a la tarea de la gestión del OS, liberando espacios de memoria asociados.
  - Pasándole como parámetro NULL elimina la tarea quien llamó a la función

```
void vOtherFunction( void )
{
     TaskHandle_t xHandle = NULL;

     // Se crea una tarea y almacena su handler.
     xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

     // Si la tarea anterior fue creada con éxito, se destruye.
     if( xHandle != NULL )
     {
         vTaskDelete( xHandle );
     }
}
```

API: **Suspensión/Reanudación de tareas**

- `void` **`vTaskSuspend`**`( TaskHandle_t xTask )`
- `void` **`vTaskResume`**`( TaskHandle_t xTask )`
  - Suspende/Reanuda la tarea, quitándola/agregándola al scheduler, sin que se altere el contexto asociado a ésta.
  - Pasándole como parámetro NULL suspende la tarea quien llamó a la función

```
void vOtherFunction( void )
{
    TaskHandle_t xHandle = NULL;

    /* Se crea una tarea y almacena su handler. */
    xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &xHandle );

    /* Si la tarea anterior fue creada con éxito, se SUSPENDE. */
    if( xHandle != NULL )
    {
        xTaskToSuspend( xHandle );
        /* hacer algo sin "la interrupción" de la tarea creada */
        vTaskResume( xHandle );
    }

    xTaskToSuspend( NULL );
}
```

API: **Forzar el llamado al scheduler**

- Si se trabaja con el FreeRTOS en modo cooperativo, hay veces que se quiere lograr, a mano, un llamado al scheduler.

- **vTaskYIELD**()

  - Si la tarea llamante es la única de su prioridad, y no hay otra de mayor prioridad, la misma continuará su ejecución.

  - Si existe otra tarea de igual prioridad que la tarea llamante, y no hay otra de mayor prioridad, se cambiará el contexto a la otra tarea de igual prioridad.

Bibliografia

- https://www.freertos.org
- FreeRTOS Kernel Documentation
- Introducción a los Sistemas operativos de Tiempo Real, Alejandro Celery - 2014
- Introducción a los Sistemas Operativos de Tiempo Real, Pablo Ridolfi, UTN, 2015.
- Introducción a Planificación de Tareas, CAPSE, Franco Bucafusco, 2017
- Introducción a Sistemas cooperativos, CAPSE, Franco Bucafusco, 2017
- FreeRTOS - Temporización, Cursos INET, Franco Bucafusco, 2017
- Rate-monotonic scheduling, Wikipedia Consultado 19-5-2
- Earliest Deadline First, Wikipedia Consultado 19-5-2

Licencia

"Introducción a los RTOS"