Software en Tiempo Real

MSc. Ing. Carlos Centeno
Ingeniería Electrónica
UTN FRC

Año 2023

Temario

- Generalidad.
- Requisitos de Hardware.
- Opciones Disponibles
- Tareas
  - TCB
  - Estados Definidos
  - Prioridades
- Sincronización con Eventos
  - Semáforos
  - Mailbox
  - Queues
  - Deadlock
- Cambio de Contexto

RTOS - Kernel

- Conjunto de funciones que llevan a cabo todas las capacidades de un RTOS.
- En términos generales:
  - Se encarga del cambio entre las tareas.
  - Se encarga de la comunicación entre las tareas.
  - Se encarga del manejo de semáforos, mailboxes, colas, demoras de tiempo.
- Necesita de ROM, RAM y tiempo de ejecución adicional para trabajar.

RTOS - Scheduler

- Es el encargado de determinar que tarea es la que se debe ejecutar.
- Necesita de la correcta distribución de prioridades para operar.
- Se ejecuta cada vez que se produce una interrupción del sistema.
  - ISRTick();
- Se ejecuta cuando finaliza un servicio del kernel.

RTOS – Tipos de Kernel

- De acuerdo a la forma en que el scheduler administra el uso del CPU se dice que existen distintos tipos de Kernel.
- Predominan dos formas específicas
- Formas de trabajo del sistema:
  - PREEMPTIVE
  - Hard Real Time
  - NON PREEMPTIVE
  - Soft Real Time

RTOS – Tipos de Kernel

- La diferencia entre Hard y Soft Real-time es la tolerancia a no cumplir con los tiempos establecidos,
  - Esto pueda llevar a una falla catastrófica.

- Para sistemas **Hard Real Time**, NO ES OPCION no cumplir con los tiempos.

- En sistema **Soft Real Time** no es crítico el no cumplir con los plazos.

RTOS – Tipos de Kernel

Hard Real Time

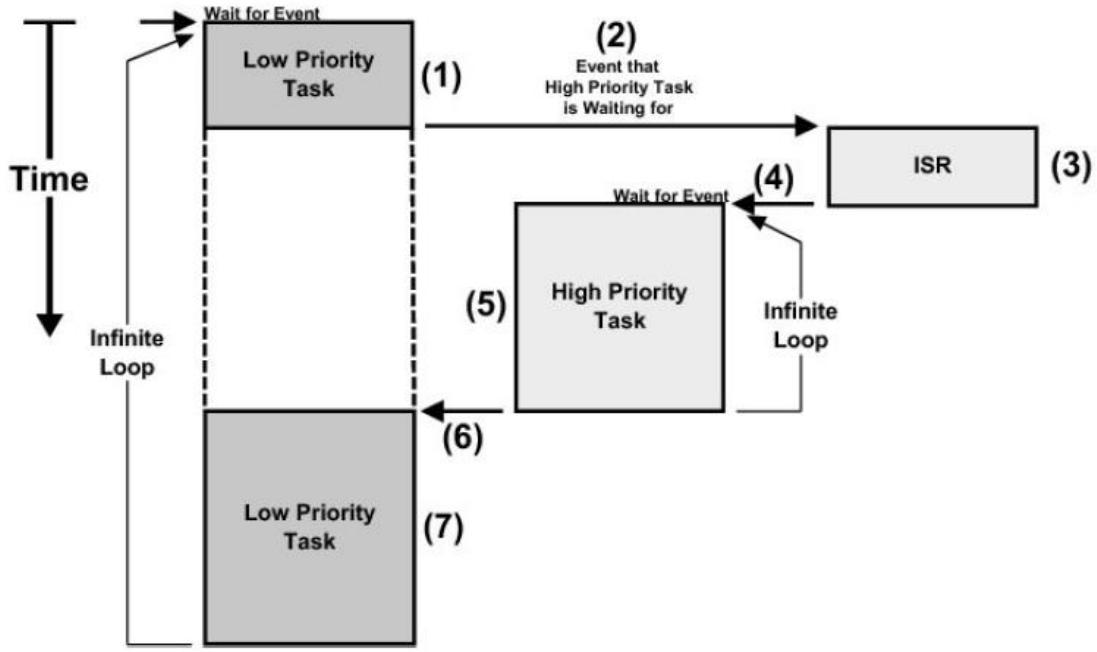
- Plazo de respuesta ESTRICTO.
  - Preemptive.
- Comportamiento temporal determinado por el diseño del sistema.

Soft Real Time

- Plazo de respuesta FLEXIBLE.
  - NON Preemtive
- Comportamiento temporal determinado por el computador.

RTOS – Preemptive

- La TAREA de mas alta prioridad SIEMPRE se ejecuta cuando es necesario.

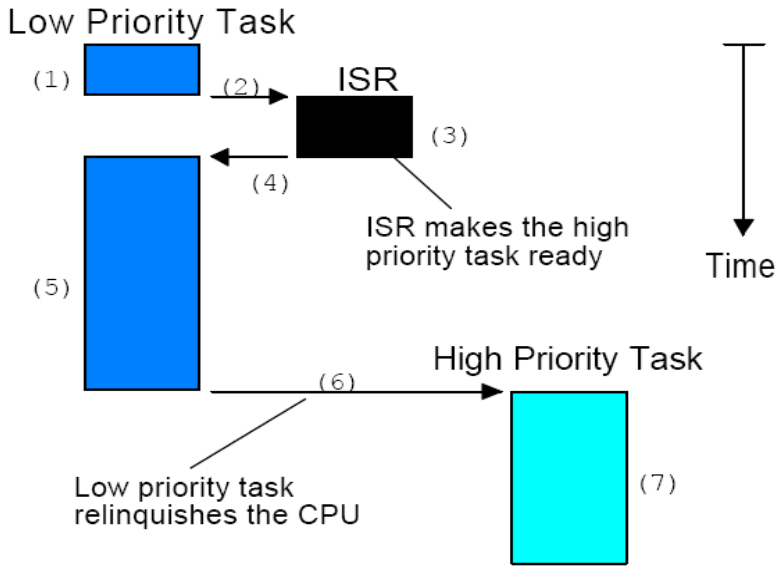


- NOTA: Se deben usar funciones reentrantes.

RTOS – NonPreemptive

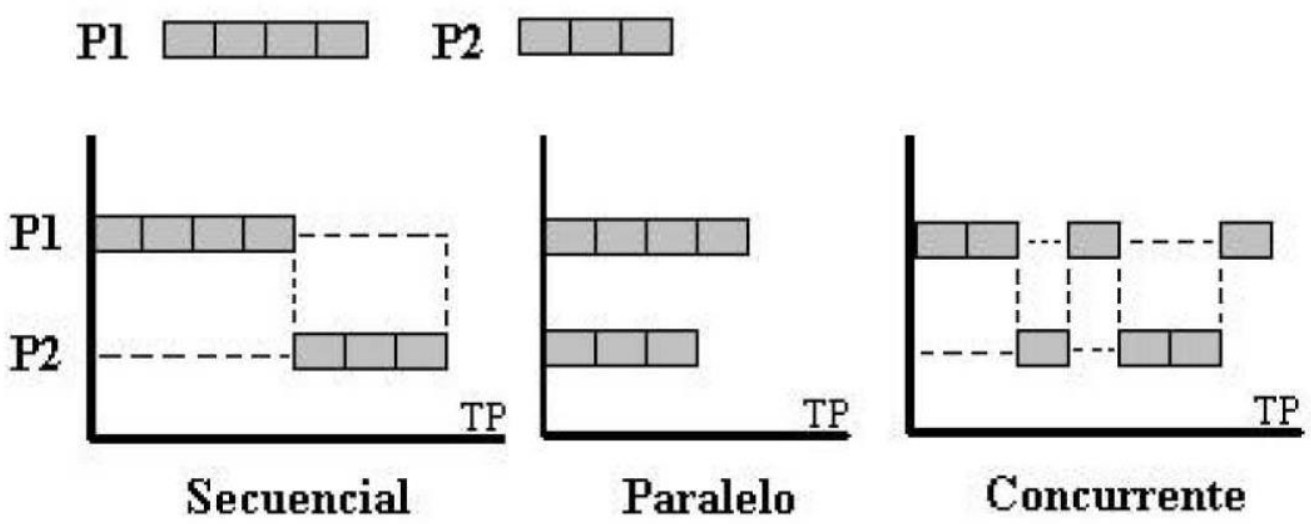
- ISR → Tarea de Mayor prioridad
- Se vuelve a la TAREA interrumpida.
- La de mas alta prioridad "READY" solo se ejecuta cuando la "ACTUAL" termina su ejecución.

- NOTA: Usar funciones reentrantes.



RTOS – Multitarea

- Se busca que el tiempo de procesamiento repartido entre las tareas (hilos) cree la ilusión de procesamiento **concurrente**.

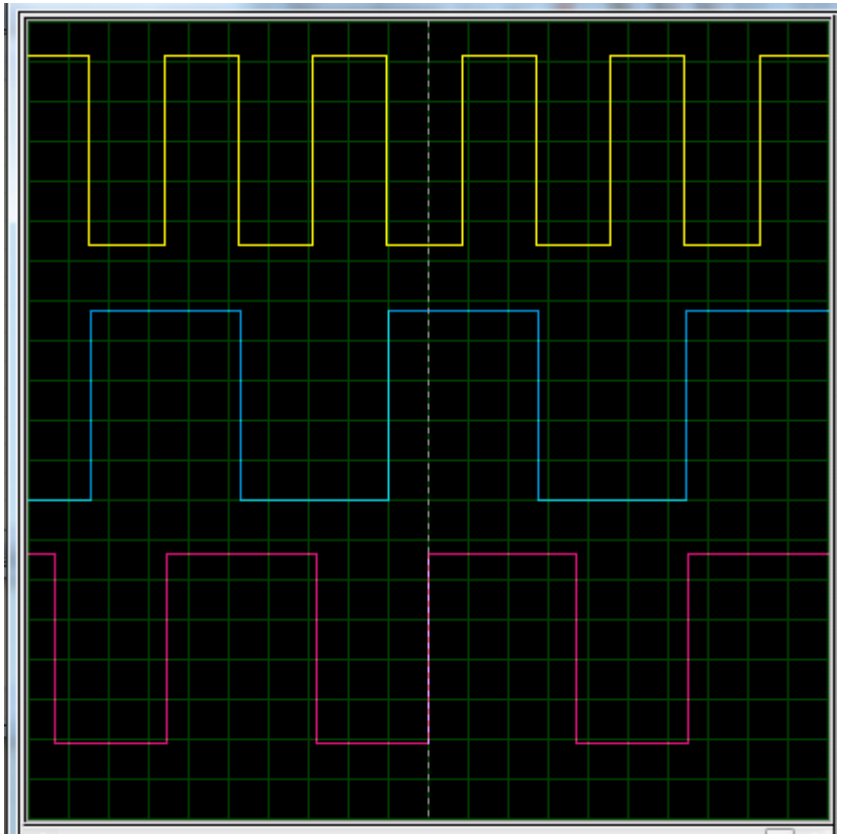


Ejemplo de Diseño

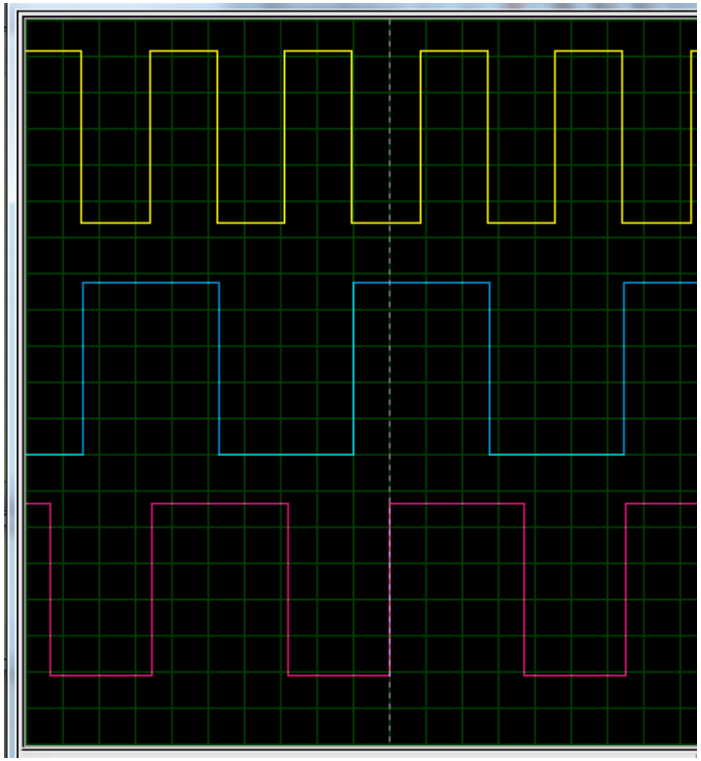
- Implementar un Sistema Embebido que controle tres secuencias temporales en salidas digitales.
- Usar topología Super Loop.
- El control de tiempo se realiza con espera pasiva.

Secuencia

- Secuencia 1:
  - Alto : 1mS
  - Bajo : 1mS
- Secuencia 2:
  - Alto : 2mS
  - Bajo : 2mS
- Secuencia 3:
  - Alto : 3mS
  - BAjo: 4mS



Secuencia Solución



```
    // SUPER loop
    for(;;){
        switch(cont1){
            case 2: salidaLed1 = 1;
                    break;
            case 4: salidaLed1 = 0;
                    cont1 = 0;
                    break;
        }
        switch(cont2){
            case 4: salidaLed2 = 1;
                    break;
            case 8: salidaLed2 = 0;
                    cont2 = 0;
                    break;
        }
        switch(cont3){
            case 3: salidaLed3 = 1;
                    break;
            case 7: salidaLed3 = 0;
                    cont3 = 0;
                    break;
        }
        Delay10TCYx(20L);
        cont1++;
        cont2++;
        cont3++;
    }
```

Espera

Solución - RTOS

- Se deben conocer aspectos específicos de RTOS.
  - Secciones Críticas
  - Reentrancia
  - Tareas
  - Estado de las Tareas
  - Bloque de control de las tareas
  - Inicialización
  - Arranque del kernel

Secciones Críticas

Secciones Críticas

- Se pueden definir secciones que denominaremos “críticas” en donde se trabaja con memoria, registros, etc.
- El uso de estos identificadores logran evitar que se produzca durante esta operación alguna interrupción ya sea por software o por hardware.

Secciones Críticas

- OS_ENTER_CRITCAL();
- OS_EXIT_CRITICAL();

- A nivel de código se definen como

```
#define  OS_CRITICAL_METHOD    2

#if      OS_CRITICAL_METHOD == 1
#define  OS_ENTER_CRITICAL()  asm  CLI
        /* Disable interrupts                    */
#define  OS_EXIT_CRITICAL()   asm  STI
        /* Enable  interrupts                    */
#endif

#if      OS_CRITICAL_METHOD == 2
#define  OS_ENTER_CRITICAL()  asm {PUSHF; CLI}
        /* Disable interrupts                    */
#define  OS_EXIT_CRITICAL()   asm  POPF
        /* Enable  interrupts                    */
#endif
```

- // Estas estan DEFINIDAS EN OS_CPU.H

Secciones Críticas

- Ejemplo de uso de Secciones Críticas

```
OS_ENTER_CRITICAL();

PC_VectSet(0x08, OSTickISR);
/* Install uC/OS-II's clock tick ISR       */

PC_SetTickRate(OS_TICKS_PER_SEC);
/* Reprogram tick rate                     */

OS_EXIT_CRITICAL();
```

Reentrancia

RTOS - Reentrancia

- Para trabajar en forma segura en un entorno multitarea las funciones deben ser reentrantes.

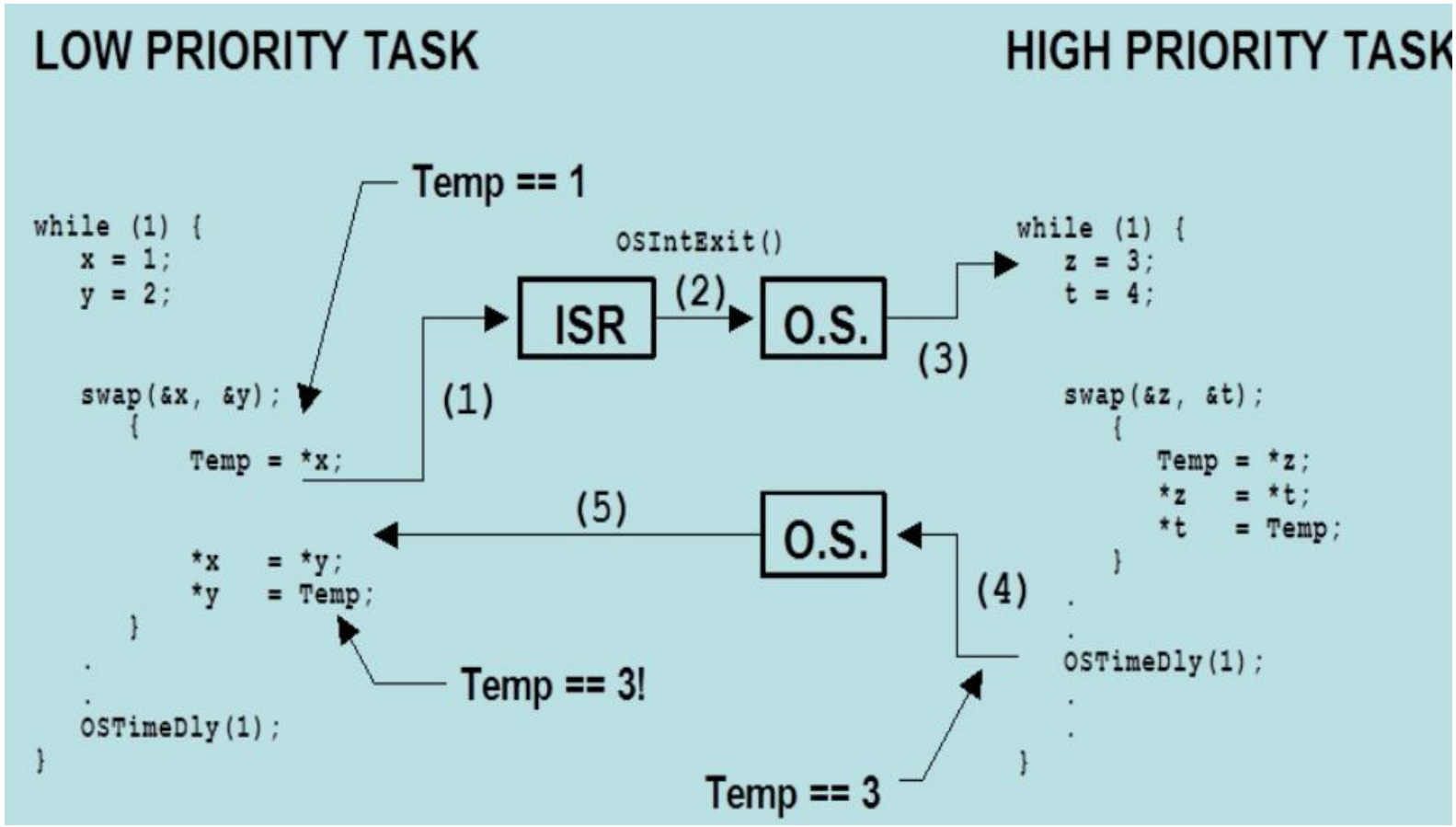
```
void strcpy(char *dest, char *src)
{
     while (*dest++ = *src++) {
          ;
     }
     *dest = NUL;
}
```

- En el ejemplo se verifica la no reentrancia de la función swap().

```
int Temp;

void swap(int *x, int *y)
{
     Temp = *x;
     *x   = *y;
     *y   = Temp;
}
```

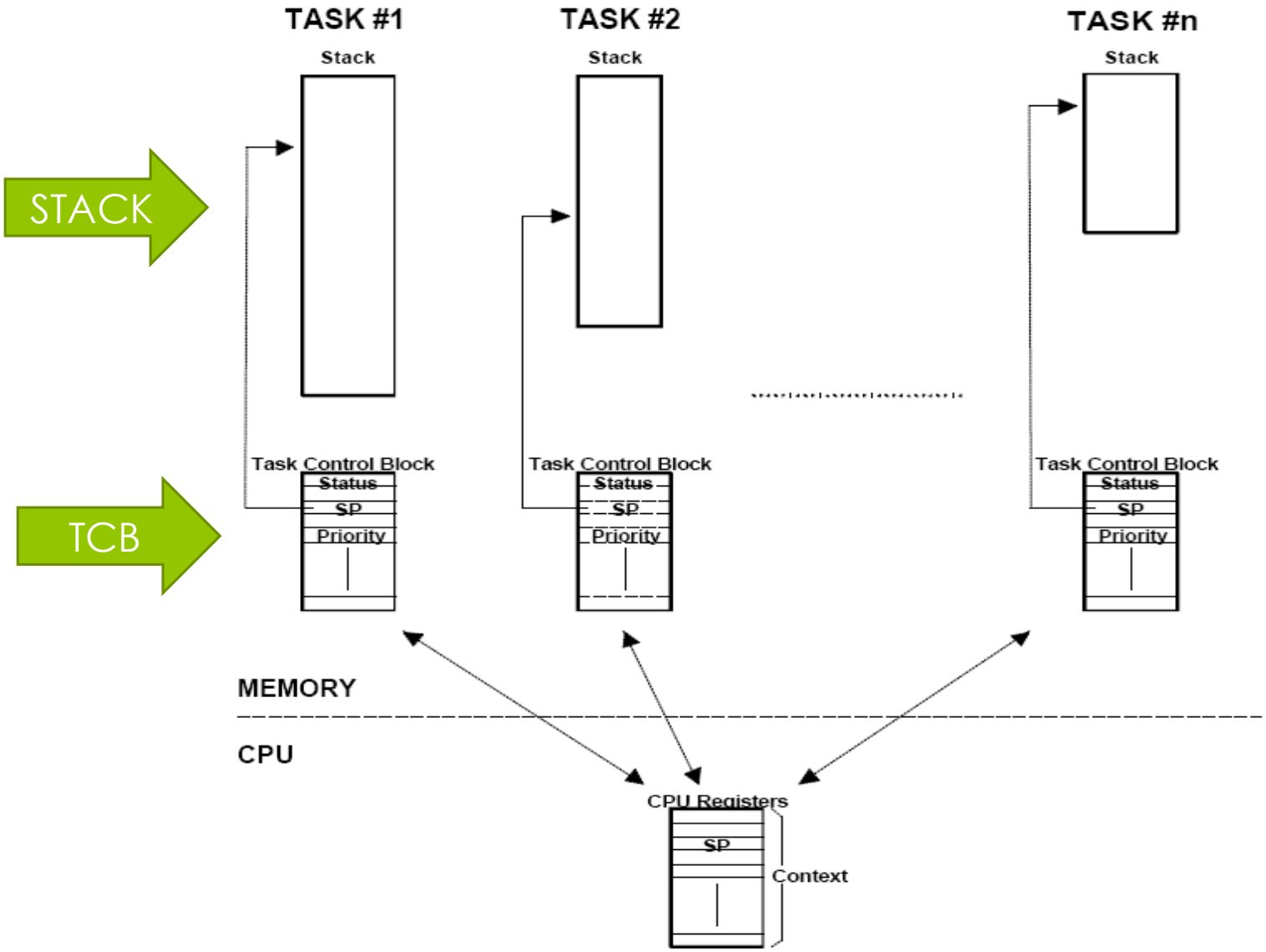
RTOS - Reentrancia



Tareas

RTOS - Tareas

- Se llama HILO (Task - Thread)
- Se divide el trabajo en bloques, que resuelven una porción del problema.
- Cada Tarea tendrá asociado:
  - TCB(Task Control Block)
  - STACK
  - Prioridad
- Se deben conocer los estados posibles.
- Se deben conocer las funcionalidades asociadas.
  - Crear. Suspender. Reasumir. Borrar. Cambio de Prioridades


TASK #1
TASK #2
TASK #n
Stack
Stack
Stack
STACK
TCB
Task Control Block
Status
SP
Priority
Task Control Block
Status
SP
Priority
Task Control Block
Status
SP
Priority
MEMORY
CPU
CPU Registers
SP
Context