Sácale provecho a tu ESP32: Introducción al sistema operativo de tiempo real FreeRTOS

Jorge Iván Marín Hurtado
Programa de Ingeniería Electrónica

UNIVERSIDAD DEL QUINDÍO

ARDUINO DAYS 2025

March 21st - 22nd

days.arduino.cc

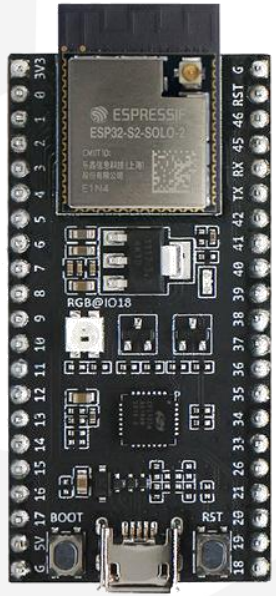
#ArduinoDays25

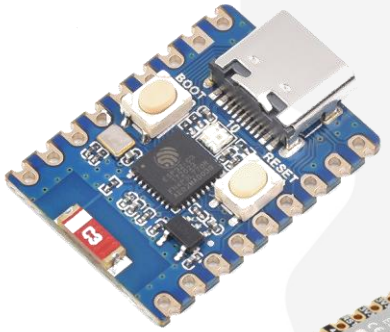
Contenido

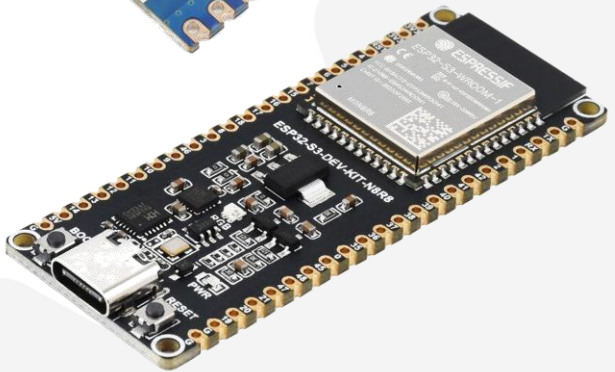
- ¿Qué es un sistema en tiempo real?
- Definiciones: tarea, *deadline*, concurrencia, tick, sección crítica
- Gestión de tareas
- Comunicación entre tareas: Colas (*Queues*)
- Semáforos
- Sección crítica y *Mutex*

¿Por qué necesitamos un RTOS?

Tarjeta	Core
ESP32-S2	Xtensa single-core 32bit LX7
ESP32-C3	32-bit single-core RISC-V
ESP32-S3	Xtensa **dual-core** 32bit LX7



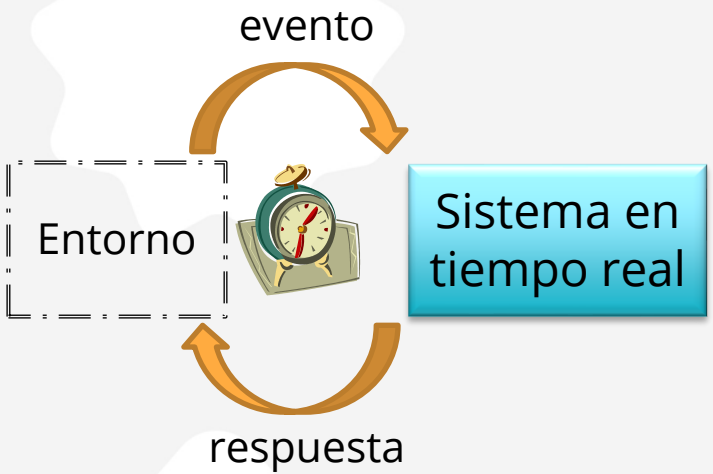




ARDUINO DAYS 2025

UNIVERSIDAD DEL QUINDÍO

¿Qué es un Sistema en Tiempo Real?



Acciones de control deben producirse en intervalos de tiempo definidos:

- Respuestas correctas dentro de un intervalo definido (*deadline*)
- Sí tiempo de respuesta > límite ➔ degradación o funcionamiento erróneo

Sistema en Tiempo Real ≠ Sistema de Alta Velocidad

ARDUINO DAYS 2025

UNIVERSIDAD DEL QUINDÍO

Aplicaciones de los RTS

Dominio Industrial

- Controlador de la planta.
- Robot para tratamiento de material peligroso.

Uso militar

- Sistema de reconocimiento de blancos automático.
- Sistema de guiado de misiles y navegación.

Sistemas altamente críticos

- Plantas nucleares
- Sistemas de aviónica

Sistemas de Telecomunicaciones y Entretenimiento

- Sistemas de Telefonía Móvil
- Sistemas Multimedia

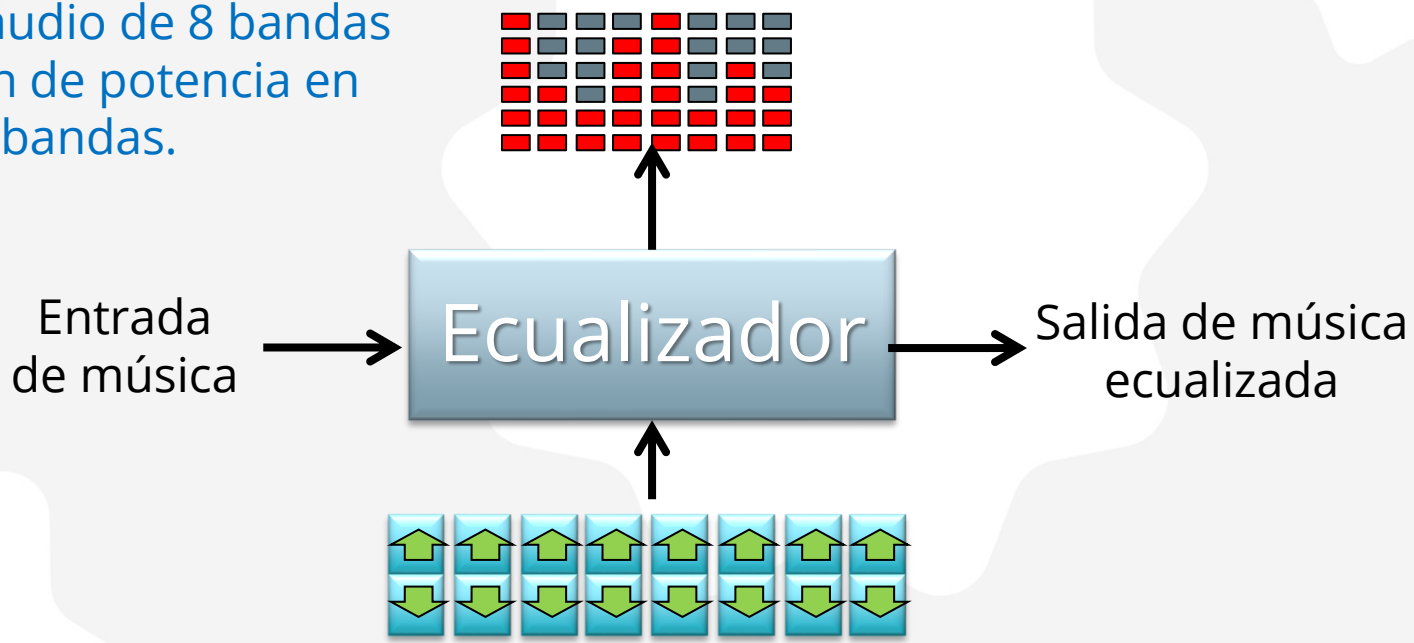
Imágenes tomadas de pixabay.com

ARDUINO DAYS 2025

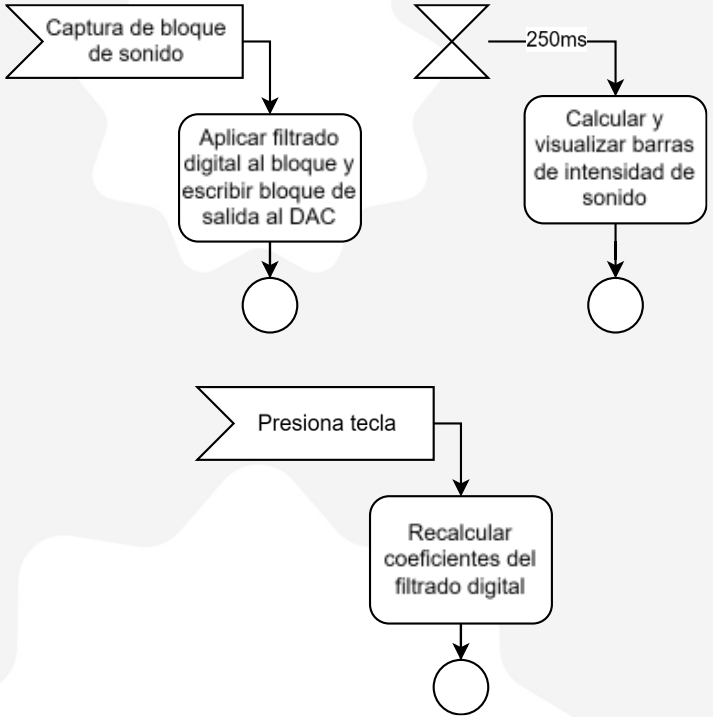
UNIVERSIDAD DEL QUINDÍO

¿Por qué es necesario un RTOS?

Caso de Estudio #1: Sistema ecualizador de audio de 8 bandas con visualización de potencia en cada una de las bandas.



¿Por qué es necesario un RTOS?



Eventos:

1. Fin captura bloque de señal de audio por DMA
   - Procesar bloque usando filtrado digital
   - Escribir bloque procesado al DAC por medio de DMA
2. Activación del teclado
   - Recalcular coeficientes de los filtros del ecualizador
3. Cada 250ms:
   - actualizar barras de nivel usando la información del bloque recién capturado.

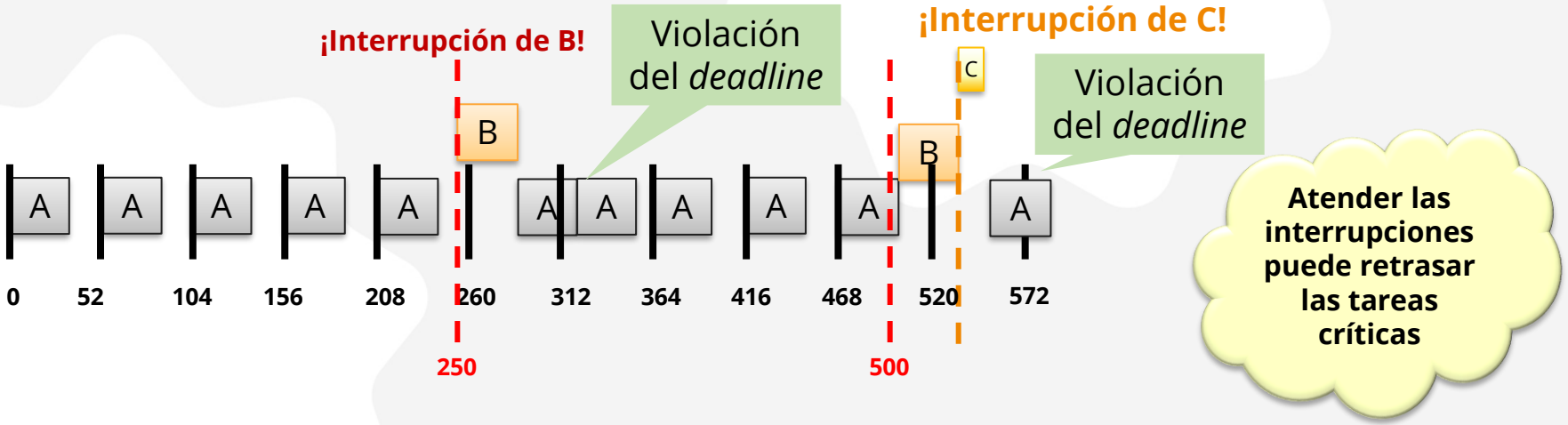
Tareas críticas: Filtrado digital

¿Por qué es necesario un RTOS?

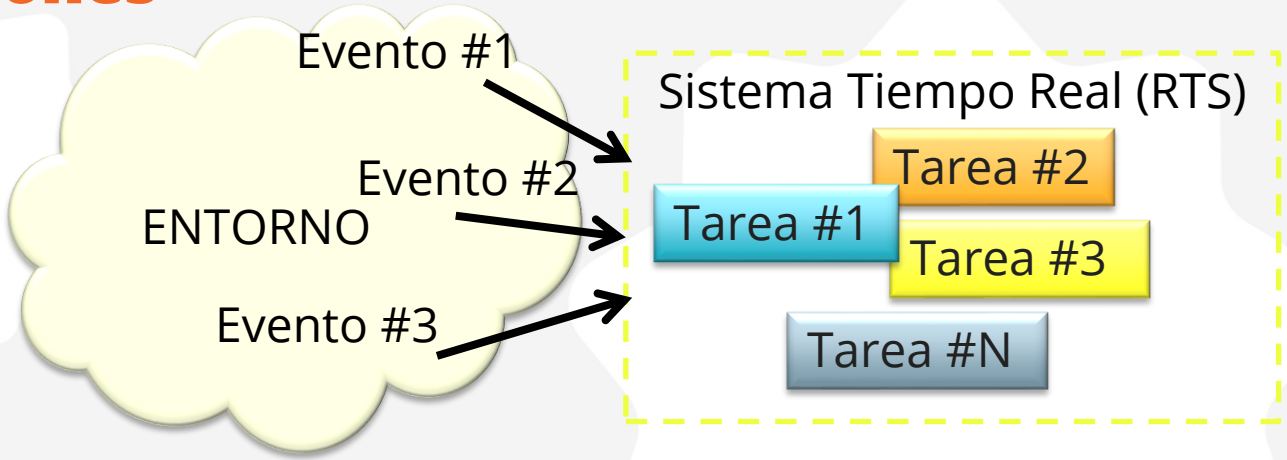
Solución Sin RTOS, usando timers e interrupciones para DMA y teclado

- Filtrado digital: Por interrupción cada 52ms (fs = 20kHz @ bloque 1000 muestras)
- Visualización cada 250ms
- Teclado: Por interrupción

Tarea	Periodicidad (T) (ms)	Duración (ms)
A: Filtrado	52	30
B: Visualización	250	30
C: Teclado	-	5



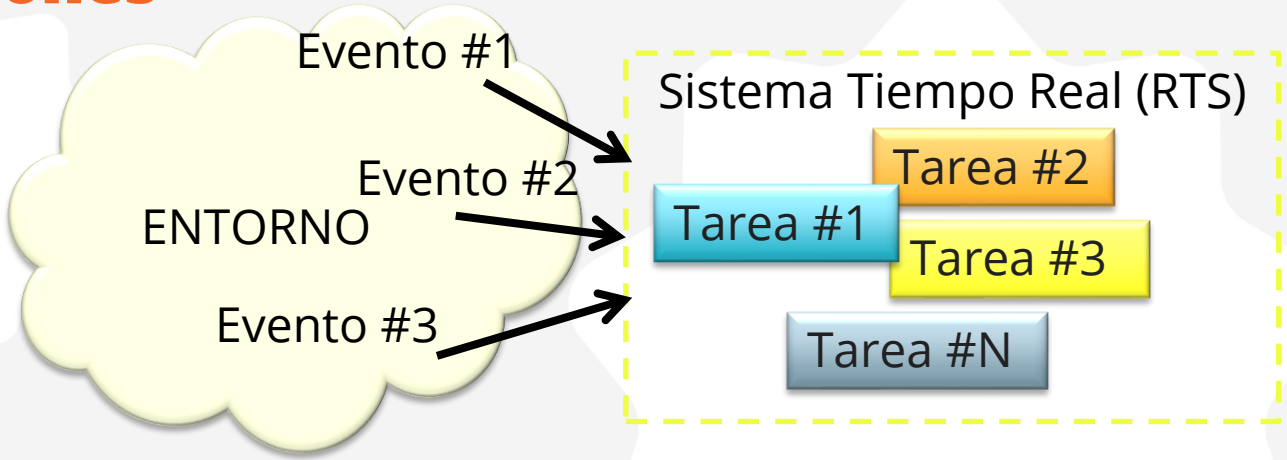
Definiciones



Eventos
- Sincrónicos → Predecibles
- Asincrónicos → No predecibles

Para un evento “debe” existir un tiempo límite de respuesta (*deadline*)

Definiciones



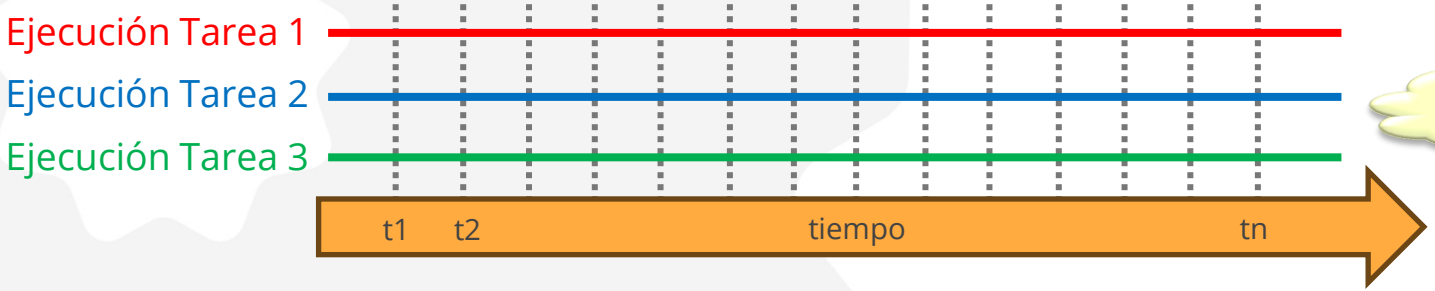
Tarea de Tiempo Real

Tareas

- Con requisitos temporales → Se satisfacen *deadlines* } Ejm: Algoritmo control
- Sin requisitos temporales → Se minimiza tiempo promedio de respuesta } Ejm: Visualización

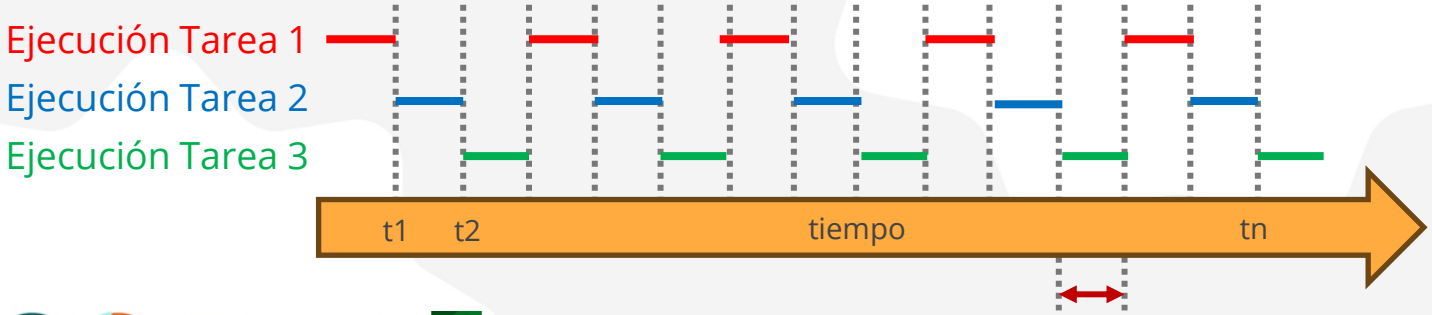
Definiciones

Todas las tareas pareciera que se ejecutaran al tiempo...



Concurrencia

...pero solo una tarea se ejecuta realmente al tiempo.



Tick

Estructura de una Tarea

setup y **loop** en el API de Arduino

No es la regla general en los sistemas embebidos

```
void setup()
{
}

void loop()
{
}
```

setup y **loop** son una abstracción de una tarea

```
void main() {
 setup();
 while(1) {
   loop();
 }
}
```

En sistemas concurrentes las tareas tienen una estructura similar (setup y loop infinito)

```
void tarea1() {
 setup1();
 while(1) {
   loop1();
 }
}
```

```
void tarea2() {
 setup2();
 while(1) {
   loop2();
 }
}
```

ESP32 soporta concurrencia a través del Sistema Operativo freeRTOS

Estructura de programación de un RTS

Creación de tareas:

```
void main()
{
   crear_tarea( CPU_Core1, control_presion );
   crear_tarea( CPU_Core2, control_temperatura );
   crear_tarea( CPU_Core1, visualizacion );
…
}
```

Tareas:

```
void control_presion()
{
    /* Inicialización */
   while(1) {
    /* Código del
       controlador */
   }
}
```

```
void control_temperatura()
{
    /* Inicialización */
   while(1) {
    /* Código del
       controlador */
   }
}
```

```
void visualizacion()
{
    /* Inicialización */
   while(1) {
    /* Código del
      visualizador */
   }
}
```

Gestión de Tareas en freeRTOS

Creación de tareas:

El *Scheduler* la asigna a un *Core*

```
xTaskCreate(
  TaskFunction, // Función con la tarea
  "Taskname", // Nombre de la tarea
  1024, // Tamaño del Stack (pila)
  NULL, // pvParameters que se pasan
        //a la función
  1, // Prioridad
  NULL // TaskHandle de retorno
);
```

Prioridad 0:
La más baja

El Usuario asigna explícitamente el *Core*

```
xTaskCreatePinnedToCore(
  TaskFunction, // Función con la tarea
  "Taskname", // Nombre de la tarea
  1024, // Tamaño del Stack (pila)
  NULL, // pvParameters que se pasan
        //a la función
  1, // Prioridad
  NULL, // TaskHandle de retorno
  1 //CPU Core
);
```

Retornan un objeto `TaskHandle_t`

Gestión de Tareas en freeRTOS

Función	Descripción
`vTaskDelete(TaskHandle_t task)`	Borra una tarea específica
`vTaskDelete(NULL)`	Borra la tarea actual
`vTaskSuspend(TaskHandle_t task)`	Suspende una tarea específica
`vTaskSuspend(NULL)`	Suspende la tarea actual
`vTaskResume(TaskHandle_t task)`	Despierta una tarea suspendida
`vTaskDelay( nTicks )`	Retardo de `nTicks`
`vTaskDelayUntil(TickType_t* lastTime, nTicks)`	Retardo de `nTicks` desde el último tiempo. Se usa en conjunto con `xTaskGetTickCount()`

Gestión de Tareas en freeRTOS

Estudio de Caso #2 – Luz parpadeante con botón start/stop

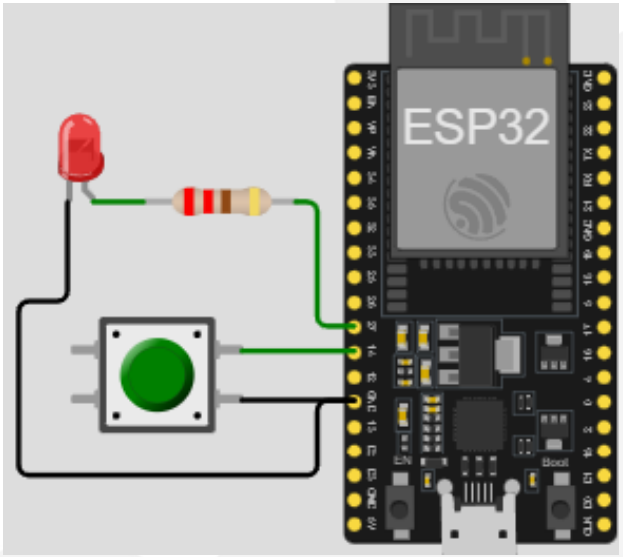
Tarea #1
Parpadea LED cada 500ms

Tarea #2
Cambia estado de tarea #1 (suspendida/ corriendo)

```
TaskHandle_t TaskHandleBlinking;

void setup() {
   xTaskCreate(
    TaskBlink, "Blink",
    1024, NULL, //Stack
    2, //Prioridad 2 (Alta)
    &TaskHandleBlinking );

   xTaskCreate(
    TaskButton, "Button",
    1024, NULL, //Stack
    1, //Prioridad 1 (Baja)
    NULL);
}
```



https://wokwi.com/projects/397136639772588033

ARDUINO DAYS 2025

UNIVERSIDAD DEL QUINDÍO

Gestión de Tareas en freeRTOS

Tarea #1
Parpadea LED cada 500ms

```
void TaskBlink(void *pvParameters) {
bool led_status = false;
pinMode(LED_OUT, OUTPUT);
  while(1) {
    digitalWrite(LED_OUT, led_status);
    led_status = !led_status;
    vTaskDelay( pdMS_TO_TICKS(500) );
  }
}
```

No usar delay() sino vTaskDelay() para permitir concurrencia.
Macro convierte tiempo en ms a ticks

Tarea #2
Cambia estado de tarea #1 (suspendida/ corriendo)

```
void TaskButton(void *pvParameters) {
  bool blinking = true;
  pinMode(BUTTON, INPUT_PULLUP);
  while(1) {
    if (digitalRead(BUTTON)==LOW) {
      blinking = !blinking;
      if (blinking==false) {
        //Suspende la tarea del LED
        vTaskSuspend(TaskHandleBlinking);
      } else {
        //Continua la tarea del LED
        vTaskResume(TaskHandleBlinking);
      }
    }
    vTaskDelay( pdMS_TO_TICKS(50) );
  }
}
```

Gestión de Tareas en freeRTOS

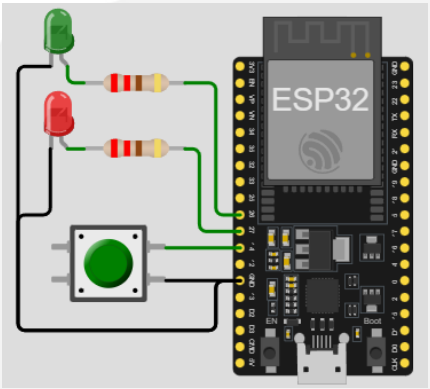
Se pueden crear múltiples instancias de una tarea usando diferente configuración a través del parámetro pvParameters

En setup():

```
xTaskCreate(
    TaskBlink, "Blink1",
    1024, (void*)27, //Parameter
    3, //Prioridad 2 (Alta)
    &TaskHandleBlinking1
  );
xTaskCreate(
    TaskBlink, "Blink2",
    1024, (void*)26, //Parameter
    2, //Prioridad 2 (Alta)
    &TaskHandleBlinking2);
```

```
void TaskBlink(void *pvParameters) {
bool led_status = false;
int pin = (int) pvParameters;
pinMode(pin, OUTPUT);
  while(1) {
    digitalWrite(pin, led_status);
    led_status = !led_status;
    vTaskDelay( pdMS_TO_TICKS(500) );
  }
}
```

El # pin está en pvParameters. Este formato permite usar estructuras para inicialización



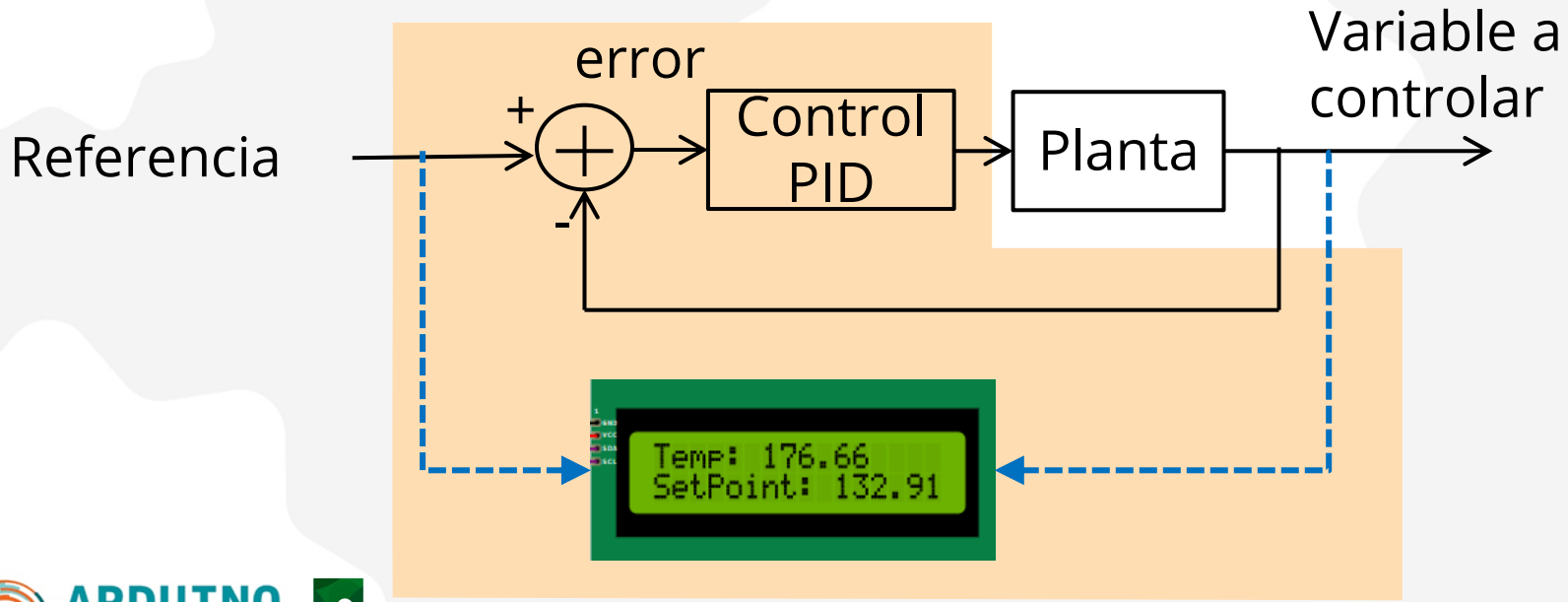
https://wokwi.com/projects/397255233279026177

ARDUINO DAYS 2025

UNIVERSIDAD DEL QUINDÍO

Gestión de Tareas en freeRTOS

- **Estudio de Caso #3**. Controlador PID para temperatura de planta con visualización de la lectura actual y la referencia (Set Point).



Gestión de Tareas en freeRTOS

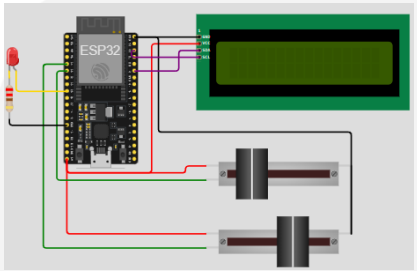
Tarea #1
Ley de control cada Ts=1,3s

Tarea #2
Visualización en Display

Rutinas LCD consumen mucho stack: Aumentarlo a 2048

```
void TaskController(void *pvParameters) {
 while(1) {
  r = analogRead(ADC_REF);
  y = analogRead(ADC_SENS);
  e = r-y;
  u = u1 + (Kp + Ki*Ts)*e + Kp*e1;
  u1 = u;
  e1 = e;
  dacWrite(DAC1, u);
  // Espera el tiempo de muestreo
  vTaskDelay( pdMS_TO_TICKS(1300) );
 }
}
```

Este delay no garantiza el Ts



https://wokwi.com/projects/397153057196163073

ARDUINO DAYS 2025

UNIVERSIDAD DEL QUINDÍO

Gestión de Tareas en freeRTOS

```
void TaskController(void *pvParameters) {
TickType_t xLastWakeTime;
while(1) {
  xLastWakeTime = xTaskGetTickCount();
  r = analogRead(ADC_REF);
  y = analogRead(ADC_SENS);
  e = r-y;
  u = u1 + (Kp + Ki*Ts)*e + Kp*e1;
  u1 = u;
  e1 = e;
  dacWrite(DAC1, u);
  // Espera el tiempo de muestreo
  vTaskDelayUntil( &xLastWakeTime, pdMS_TO_TICKS(1300) );
 }
}
```

Este delay SI garantiza el Ts independiente de lo que demoren las instrucciones

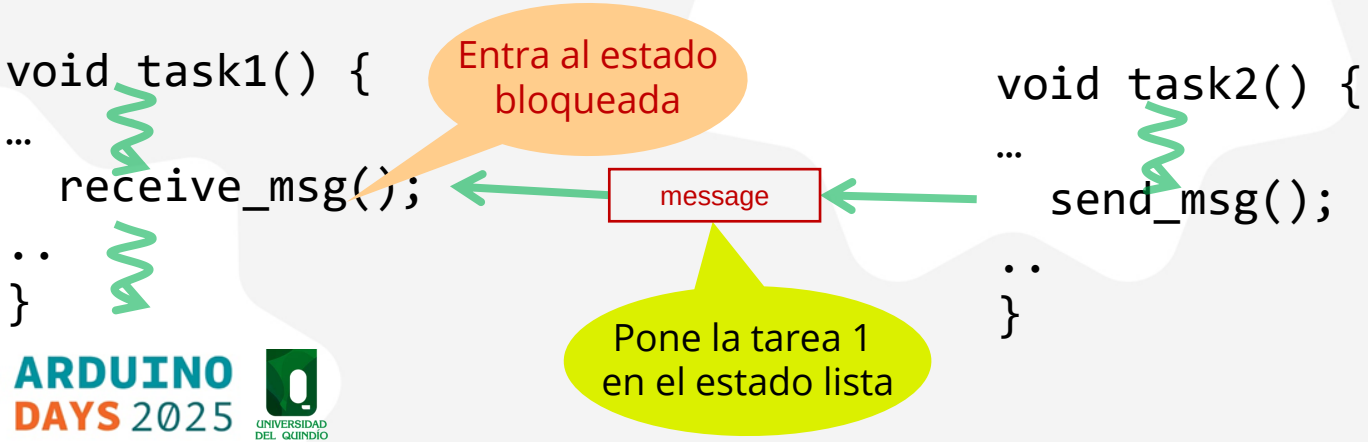
Comunicación entre tareas

En sistemas simples,

- variables globales bastan para intercambiar datos entre dos tareas
- no es conveniente cuando varias tareas acceden al mismo recurso o hay eventos en cola

Solución: Colas (*Queues*)

- Una tarea se prepara para recibir y la otra enviar
- Colas tienen comportamiento FIFO (*First In – First Out*)
- Permiten bloquear tareas hasta que llegue un dato o se complete un *timeout*



Comunicación entre tareas

- Colas en freeRTOS:

Función	Descripción
`QueueHandle_t` **`xQueueCreate`**`( QueueLength, ItemSize)`	Crea una cola con un tamaño máximo de mensajes `QueueLength` y donde cada ítem tiene un tamaño de `ItemSize` bytes
`xQueueSend``( QueueHandle_t , ItemToQueue, xTicksToWait )`	Envía un ítem a la cola. Espera `xTicksToWait` si la cola está llena. Retorna `pdTRUE` o `errQUEUE_FULL`
`xQueueReceive``( QueueHandle_t, Buffer, xTicksToWait )`	Recibe un ítem de la cola. Espera `xTicksToWait` si la cola está vacía. Retorna `pdTRUE` o `pdFALSE`

Si se usan colas desde ISR se debe usar `xQueueSendFromISR`

Comunicación entre tareas

- Ejemplo Controlador PID: Uso de colas para enviar visualizar la referencia y setpoint

```
void TaskController(void *) {
 TickType_t xLastWakeTime;
 while(1) {
  ...
  //Envía datos a las colas
  if( xQueueSens != 0 ) {
    if (xQueueSend(xQueueSens,&y,0)
       ==pdTRUE){
      // Mensaje enviado
    }
  }
  if( xQueueRef != 0 ) {
    if (xQueueSend(xQueueRef,&r,0)
       ==pdTRUE){
      // Mensaje enviado
    }
  }
  ...
```

```
void TaskDisplay(void *) {
 int measure, setpoint;
  ...
 while(1) {
  //Muestra la temperatura actual
  if(xQueueReceive(xQueueSens,&measure,10)==pdTRUE){
    lcd_1.setCursor(6, 0);
    lcd_1.print(measure/10.24); lcd_1.print("   ");
  }
  //Muestra la referencia
  if(xQueueReceive(xQueueRef,&setpoint,10)==pdTRUE){
    lcd_1.setCursor(10, 1);
    lcd_1.print(setpoint/10.24); lcd_1.print("   ");
  }
  vTaskDelay( pdMS_TO_TICKS(1000) );
 }
}
```

ARDUINO DAYS 2025 — UNIVERSIDAD DEL QUINDÍO

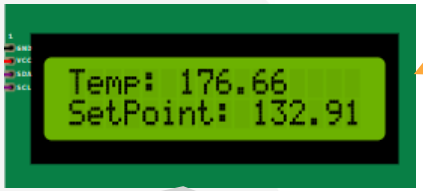
https://wokwi.com/projects/397186618654615553

Acceso a Recursos Compartidos

- ¿Qué pasa si dos tareas tratan de escribir concurrentemente al display?

Tarea #1
Escribir
Temp:180.15

Tarea #2
Escribir
SetPoint: 55.5



Se pueden producir resultados impredecibles si se le quita el control a una tarea que este usando el display

Solución: Zonas de exclusión mutua o Mutex

Acceso a Recursos Compartidos

- Para acceder una zona de exclusión mutua o *mutex* se usan semáforos:

```
SemaphoreHandle_t xSemaphore = NULL;
```
← Variable global

```
xSemaphore = xSemaphoreCreateMutex();
```
← En el setup()

```
//Revisa si el recurso compartido está libre
if( xSemaphoreTake( xSemaphore, (TickType_t) 10 ) == pdTRUE )
{
 //Código cuando el recurso compartido está libre
        ...
 //Libera el recurso compartido
 xSemaphoreGive( xSemaphore );
}
else
{
   //Código cuando no se puede acceder al recurso compartido
}
```

Esto se hace en cada tarea que use el recurso compartido

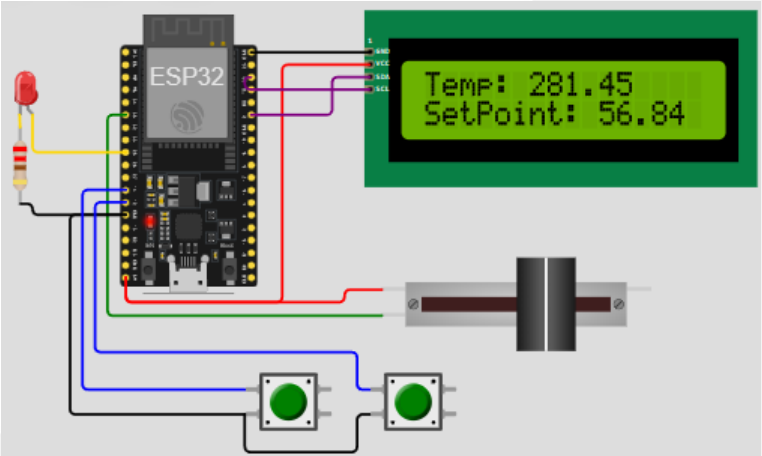
Acceso a Recursos Compartidos

- Ejemplo: Una nueva versión del controlador PID incluye dos botones (up/down) para cambiar el setpoint.

Tarea #1
Ley de control cada Ts=1,3s
- Usa cola para enviar medida de temperatura
- Usa setpoint almacenado

Tarea #2
Visualiza medida
- Usa cola para leer medida temperatura
- Escribe en display la medida

Tarea #3
Cambio y Visualización del setpoint
- Lee botones y modifica setpoint
- Escribe en display el nuevo setpoint



https://wokwi.com/projects/397190597456941057

Referencias

Multitarea en FreeRTOS para ESP32:
https://www.electrosoftcloud.com/freertos-en-esp32-esp8266-multi-tarea/

freeRTOS API Reference: https://www.freertos.org/a00106.html

Jorge Iván Marín Hurtado
jorgemarin@uniquindio.edu.co

Thank you!

March 21st - 22nd
days.arduino.cc
#ArduinoDays25