



**INSTITUTO POLITÉCNICO
NACIONAL.**

UNIDAD PROFESIONAL
INTERDISCIPLINARIA DE
INGENIERÍA CAMPUS ZACATECAS



SISTEMAS OPERATIVOS EN TIEMPO REAL.

Práctica 01,02&03 - Instalación, conociendo el entorno y creación de procesos.

Profesor: M. Ramón Jaramillo Martínez.

Miembros del equipo:

Fernando Murillo Franco

Rafael Martin Enciso Cuevas

Roberto Coronel de Santiago

Grupo: 3MM6

Carrera: Ingeniería Mecatrónica

Fecha de entrega: 25 de septiembre del 2023

ANTECEDENTES.

- ¿Qué es un sistema operativo en tiempo real?

Un sistema operativo en tiempo real (Real Time Operating System), también conocido como RTOS por sus siglas en inglés, es un sistema operativo liviano utilizado para desarrollar cosas íntimas e integrar tareas de diseño con recursos y tiempos específicos de manera óptima y sencilla como suele aplicar para los sistemas integrados.

Son aquellos en los cuales no tiene importancia el usuario, sino los procesos. Por lo general, están subutilizados sus recursos con la finalidad de prestar atención a los procesos en el momento que lo requieran. Se utilizan en entornos donde son procesados un gran número de sucesos o eventos.

Estos sistemas son esenciales en aplicaciones donde la respuesta rápida y predecible es fundamental, como en sistemas de control industrial, sistemas de navegación de vehículos, sistemas médicos, aviónica, robótica y otros entornos similares.

- ¿Cuál es la diferencia entre un sistema operativo en tiempo real contra un sistema operativo convencional?

Su principal característica es su diseño y en la forma en que gestionan las tareas y los recursos. Algunos ejemplos de ellos es que los RTOS tienen la capacidad para garantizar que las tareas críticas se ejecuten dentro de plazos estrictos y predecibles. Los sistemas operativos convencionales no pueden proporcionar esta garantía, ya que se centran en la multitarea, donde las tareas pueden ejecutarse en tiempos variables según la carga del sistema.

En un RTOS, las tareas se planifican en función de su prioridad y de los plazos de ejecución. Las tareas críticas tienen prioridad sobre las tareas menos críticas. En los sistemas operativos convencionales, las tareas se planifican de acuerdo con algoritmos de programación como el planificador de procesos, que busca un equilibrio entre la equidad y la eficiencia en lugar de garantizar plazos estrictos. También los RTOS proporcionan mecanismos de comunicación y sincronización más simples y eficientes entre tareas, ya que están diseñados para aplicaciones donde estas son críticas. Los sistemas operativos convencionales ofrecen una variedad de herramientas de comunicación, pero a menudo están optimizados para un rendimiento más generalista.

Los sistemas operativos convencionales se diseñan para proporcionar una amplia variedad de características y servicios que pueden ser utilizados por aplicaciones de propósito general, lo que les da una mayor flexibilidad. Los sistemas operativos en tiempo real se centran en proporcionar una funcionalidad mínima pero predecible para aplicaciones críticas. Además, los RTOS están diseñados para ser eficientes en el uso de recursos, ya que a menudo se ejecutan en sistemas con recursos limitados. Los sistemas operativos convencionales, por otro lado, suelen estar más enfocados en la gestión de recursos de manera eficiente en aplicaciones de propósito general.

Y como final, los sistemas operativos en tiempo real son más seguros y confiables. Resisten errores y fallos.

- Ventajas y desventajas de los sistemas operativos en tiempo real.

VENTAJAS

- Máximo consumo: Máxima utilización de dispositivos y sistemas. Por lo tanto, más rendimiento de todos los recursos.
- Cambio de tareas: el tiempo asignado para las tareas de cambio en estos sistemas es muy inferior. Por ejemplo, en sistemas más antiguos, tarda unos 10 microsegundos. Cambiar una tarea a otra y en los últimos sistemas, toma 3 microsegundos.
- Centrarse en la aplicación: centrarse en las aplicaciones en ejecución y darle menos importancia a las aplicaciones que están en la cola.
- Sistema operativo en tiempo real en el sistema integrado: dado que el tamaño de los programas es pequeño, RTOS también puede ser un sistema integrado como en el transporte y otros.
- Sin errores: estos tipos de sistemas no tienen muchos errores.
- Asignación de memoria: la asignación de memoria se administra mejor en este tipo de sistemas.

DESVENTAJAS:

- Tareas limitadas:
 - muy pocas tareas se ejecutan simultáneamente y su concentración es muy menor en pocas aplicaciones para evitar errores.
 - Use recursos pesados del sistema:
 - a veces, los recursos del sistema no son tan buenos y también son costosos.
 - Algoritmos complejos:
 - los algoritmos son muy complejos y difíciles de escribir para el diseñador.
 - Controlador de dispositivo y señales de interrupción:
 - necesita controladores de dispositivo específicos y señales de interrupción para responder lo antes posible a las interrupciones.
 - Prioridad de subprocesos: no es bueno establecer la prioridad de los subprocesos, ya que estos sistemas son menos propensos a cambiar de tarea.
 - Conmutación mínima: RTOS realiza una conmutación mínima de tareas.
- Tipos de sistemas operativos en tiempo real que hay en el mercado.

Tiempo real duro:

En estos tipos de sistemas operativos, los plazos de tiempo límite son muy estrictos lo que significa que las tareas que se realicen deben comenzar su ejecución en el tiempo programado especificado y debe completarse forzosamente dentro del periodo de tiempo que se le asigne.

Tiempo real firme:

En este tipo de sistema operativo también se debe cumplir con plazos de tiempo, sin embargo, el incumplimiento en algún plazo límite puede no ser de gran impacto por lo que existe cierta flexibilidad con el tiempo de finalización de la tarea lo puede causar efectos no deseados en el funcionamiento del sistema como la reducción de la calidad.

Tiempo real suave:

En este tipo se aceptan retrasos por parte del propio sistema operativo. Hay un periodo de tiempo límite para una tarea específica, pero son aceptables los retrasos que se puedan dar siempre y cuando el tiempo no sea muy alto, es decir se manejar retrasos suavemente.

Adicionalmente son relativamente estables, pero no muy robustos, pueden ser tolerantes a fallas, baja su desempeño con cargas excesivas de tareas por realizar, no es requisito la seguridad informática y de operación, pueden manejar grandes volúmenes de información.

Adjuntamos algunos ejemplos de sistemas operativos que hay en el mercado:

FreeRTOS: Es un RTOS de código abierto y altamente popular. Es conocido por ser liviano y adecuado para sistemas embebidos y aplicaciones de Internet de las cosas (IoT). FreeRTOS se utiliza ampliamente en una variedad de dispositivos y es ampliamente respaldado por la comunidad de desarrolladores.

RTOS de tiempo real de Nucleus (Nucleus RTOS): Desarrollado por Mentor Graphics (ahora parte de Siemens), Nucleus RTOS es un sistema operativo en tiempo real ampliamente utilizado en sistemas embebidos y dispositivos de consumo. Ofrece una amplia gama de características y es altamente configurable.

QNX: QNX es un RTOS altamente confiable y utilizado en aplicaciones críticas, como sistemas de automóviles, dispositivos médicos y equipos de telecomunicaciones. Es conocido por su alto grado de determinismo y seguridad.

VxWorks: Desarrollado por Wind River (ahora parte de Intel), VxWorks es un RTOS popular en aplicaciones aeroespaciales, militares, industriales y de telecomunicaciones. Ofrece un alto grado de confiabilidad y determinismo.

Micrium μ C/OS: Micrium μ C/OS es otro RTOS de código abierto que se utiliza en sistemas embebidos y aplicaciones críticas. Ofrece una versión gratuita llamada μ C/OS-II y una versión comercial llamada μ C/OS-III.

RTLinux: Es una extensión del sistema operativo Linux que añade capacidades en tiempo real a Linux. Es utilizado en aplicaciones donde se necesita combinar la flexibilidad de Linux con la determinismo de un RTOS.

- Arquitectura de microcontroladores que soportan un RTOS.

Entre las arquitecturas más comunes, se describen las siguientes:

ARQUITECTURA ARM:

Los microcontroladores basados en la arquitectura ARM son muy populares para la implementación de RTOS. Las familias de microcontroladores ARM Cortex-M (como Cortex-M0, Cortex-M3, Cortex-M4, y Cortex-M7) son ampliamente utilizadas en aplicaciones embebidas y son compatibles con varios RTOS. ARM Cortex-A, que se utiliza en aplicaciones más robustas, también puede ejecutar RTOS, pero se utiliza principalmente en sistemas embebidos más potentes.

ARQUITECTURA X86:

Aunque más comúnmente asociada con sistemas de PC y servidores, la arquitectura x86 también se utiliza en microcontroladores de alto rendimiento. Los microcontroladores x86 generalmente tienen más recursos y se utilizan en aplicaciones que requieren un mayor poder de procesamiento y capacidad de memoria. Estos microcontroladores pueden ejecutar RTOS, así como sistemas operativos completos como Linux o Windows Embedded.

ARQUITECTURA MIPS:

Los microcontroladores basados en la arquitectura MIPS también se utilizan en aplicaciones embebidas y pueden ejecutar RTOS. MIPS es conocido por su eficiencia en términos de energía y se utiliza en una variedad de dispositivos, incluidos enrutadores, sistemas de entretenimiento en el hogar y sistemas embebidos en automóviles.

ARQUITECTURA RISC-V:

RISC-V es una arquitectura de conjunto de instrucciones abierta y se ha vuelto cada vez más popular en la industria de microcontroladores. Los microcontroladores basados en RISC-V son altamente personalizables y pueden ejecutar RTOS como FreeRTOS, además de sistemas operativos completos. Esta arquitectura ofrece flexibilidad y es especialmente adecuada para aplicaciones personalizadas y de bajo consumo de energía.

ARQUITECTURAS PROPIETARIAS:

Algunos fabricantes de microcontroladores desarrollan arquitecturas propietarias para sus productos. Estas arquitecturas suelen ser altamente optimizadas para aplicaciones específicas y pueden ejecutar RTOS diseñados específicamente para esas plataformas.

MATERIAL NECESARIO.

- Tarjeta de desarrollo ESP32.
- Osciloscopio.
- Arduino Ide.
- Resistencias y Leds.
- Push Button.

PROCEDIMIENTO.

1. Estructura de programación.

1.1. Genere un código en el entorno de Arduino que considere los siguiente:

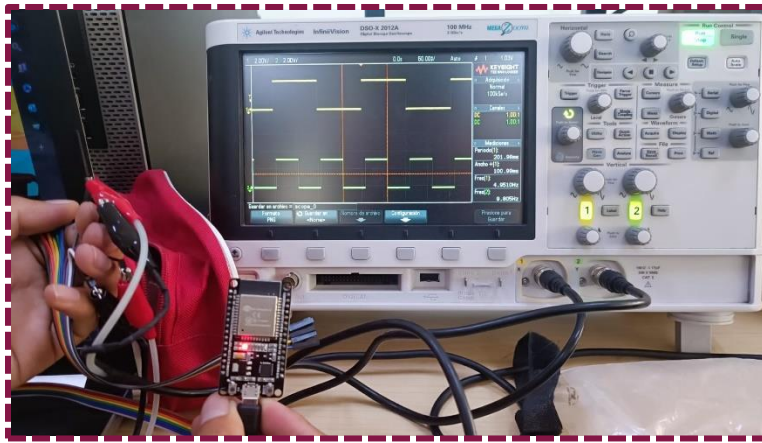
1. Toggle de dos salidas digitales (Leds).
2. La primera salida digital con un periodo de 500 ms.
3. La segunda salida digital con un periodo de 323 ms.

```
#include <Arduino.h>
// Variables
const int pinSalida1 = 12;
const int pinSalida2 = 13;

//Variables en falso para el void loop
bool estadoSalida1 = false;
bool estadoSalida2 = false;

//Inicializo en cero
unsigned long tiempoAnterior1 = 0;
unsigned long tiempoAnterior2 = 0;
const unsigned long intervalo1 = 500; // 500 ms
const unsigned long intervalo2 = 323; // 323 ms

// defino pines de salida
void setup() {
  pinMode(pinSalida1, OUTPUT);
  pinMode(pinSalida2, OUTPUT);
}
void loop() {
  //funcion para medir tiempo actual
  unsigned long tiempoActual = millis();
  // secuencia led 1
  if (tiempoActual - tiempoAnterior1 >= intervalo1) {
    estadoSalida1 = !estadoSalida1;
    digitalWrite(pinSalida1, estadoSalida1);
    tiempoAnterior1 = tiempoActual;
  }
  // secuencia led 2
  if (tiempoActual - tiempoAnterior2 >= intervalo2) {
    estadoSalida2 = !estadoSalida2;
    digitalWrite(pinSalida2, estadoSalida2);
    tiempoAnterior2 = tiempoActual;
  }
}
```

1.2. Genere un código en el entorno de Arduino que considere los siguiente:

1. Mismas características al código anterior, pero utilizando FreeRTOS.
2. Considere ambas tareas con la misma prioridad.

```
#if CONFIG_FREERTOS_UNICORE
static const BaseType_t app_cpu = 0;
#else
static const BaseType_t app_cpu = 1;
#endif

//Pins
static const int LED01 = 5;
static const int LED02= 4;

//ms
static const int rate_1= 500;
static const int rate_2= 323 ;

//Tarea 01: Toggle LED01
void toggleLED_1(void*parameter) {
    while(1) {
        digitalWrite(LED01,HIGH);
        vTaskDelay(rate_1 / portTICK_PERIOD_MS);
        digitalWrite(LED01,LOW);
        vTaskDelay(rate_1 / portTICK_PERIOD_MS);
    }
}

//Tarea 02: Toggle LED02
void toggleLED_2(void*parameter) {
    while(1) {
        digitalWrite(LED02,HIGH);
        vTaskDelay(rate_2 / portTICK_PERIOD_MS);
        digitalWrite(LED02,LOW);
        vTaskDelay(rate_2 / portTICK_PERIOD_MS);
    }
}
```

```

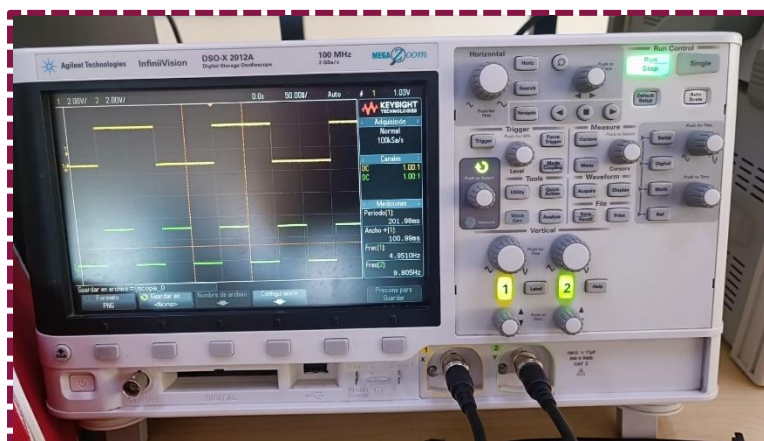
}
}

void setup() {
  pinMode(LED01, OUTPUT);
  pinMode(LED02, OUTPUT);
  xTaskCreatePinnedToCore (
    toggleLED_1,    // Funcion que se manda a llamar
    "Toggle 01",    // El nombre de la tarea
    1024,           // Stack size
    NULL,           // p
    1,              // Prioridades
    NULL,           //
    app_cpu);       //Core
  // put your setup code here, to run once:

  xTaskCreatePinnedToCore (
    toggleLED_2,    // Funcion que se manda a llamar
    "Toggle 02",    // El nombre de la tarea
    1024,           // Stack size
    NULL,           //
    1,              // Prioridades
    NULL,           //
    app_cpu);       //Core
  // put your setup code here, to run once:
  // NOTA: Si queremos actividades muy especificas que sucedan al mismo tiempo podemos
  observar un desfase esto es por que el micro no es multitarea
}

void loop() {
  // put your main code here, to run repeatedly:
}

```



1.3. Realice las siguientes modificaciones a los códigos.

1. Integre un botón, el cual tendrá como función modificar el periodo. Al presionar el botón, la primera salida cambiará su periodo a 1 segundo y la segunda salida cambiará su periodo a 500 ms. Revise en el osciloscopio ambas señales.

```
// Solo se utiliza el core 1
#if CONFIG_FREERTOS_UNICORE
static const BaseType_t app_cpu = 0;
#else
static const BaseType_t app_cpu = 1;
#endif

//Pins
int static const LED01 = 5;
int static const LED02 = 4;
int static const SW1 = 18;
int status = 0;

//Tarea 01: Toggle LED01
void TareaLED01(void* parameter)
{
    while (1)
    {
        status = digitalRead(SW1);
        if (status == HIGH)
        {
            digitalWrite(LED01, HIGH);
            vTaskDelay(500 / portTICK_PERIOD_MS);
            digitalWrite(LED01, LOW);
            vTaskDelay(500 / portTICK_PERIOD_MS);
        }
        else
        {
            digitalWrite(LED01, HIGH);
            vTaskDelay(1000 / portTICK_PERIOD_MS);
            digitalWrite(LED01, LOW);
            vTaskDelay(1000 / portTICK_PERIOD_MS);
        }
    }
}

//Tarea 02: Toggle LED02
void TareaLED02(void* parameter)
{
    while (1)
    {
        status = digitalRead(SW1);
        if (status == HIGH)
```

```

{
    digitalWrite(LED02, HIGH);
    vTaskDelay(323 / portTICK_PERIOD_MS);
    digitalWrite(LED02, LOW);
    vTaskDelay(323 / portTICK_PERIOD_MS);
}
else
{
    digitalWrite(LED02, HIGH);
    vTaskDelay(500 / portTICK_PERIOD_MS);
    digitalWrite(LED02, LOW);
    vTaskDelay(500 / portTICK_PERIOD_MS);
}
}
}

void setup()
{
    pinMode(SW1, INPUT);
    pinMode(LED01, OUTPUT);
    pinMode(LED02, OUTPUT);
    xTaskCreatePinnedToCore(
        TareaLED01, // Funcion que se manda a llamar
        "LED 01",  // El nombre de la tarea
        1024,      // Stack size
        NULL,      //
        1,         // Prioridades
        NULL,      //
        app_cpu);  //Core
                // put your setup code here, to run once:

    xTaskCreatePinnedToCore(
        TareaLED02, // Funcion que se manda a llamar
        "LED 02",  // El nombre de la tarea
        1024,      // Stack size
        NULL,      //
        1,         // Prioridades
        NULL,      //
        app_cpu);  //Core
        // put your setup code here, to run once:
        // NOTA: Si queremos actividades muy especificas que sucedan al mismo tiempo podemos
        observar un desfase esto es por que el micro no es multitarea
    }

void loop()
{
    // put your main code here, to run repeatedly:
}

```

2. ¿Es posible diferenciar cuál de las dos señales se habilita primero?

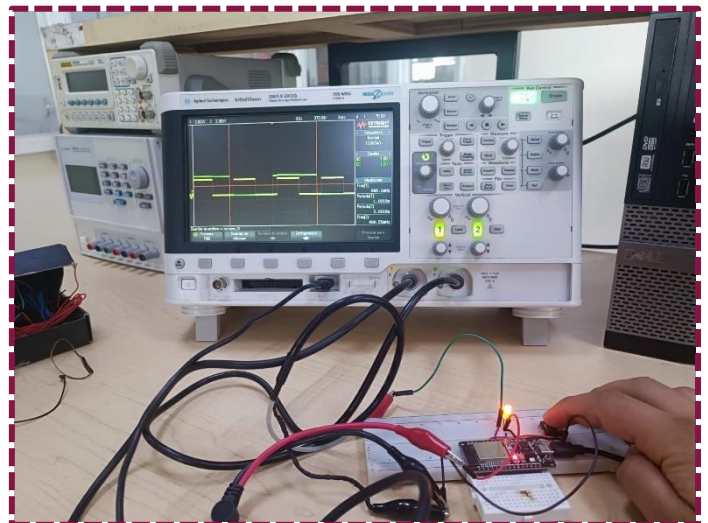
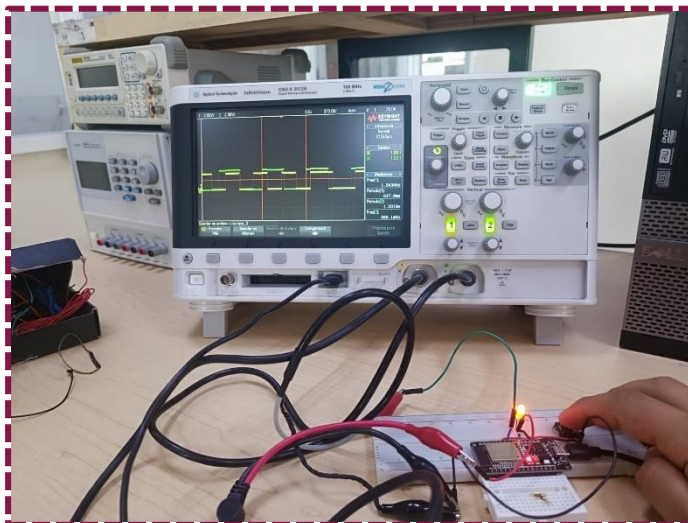
Ambas señales parecen ser habilitadas al mismo tiempo y por su velocidad no podemos observar cual es la que empieza primero esto gracias a que las tareas tienen la misma prioridad.

3. ¿El periodo de trabajo corresponde al deseado?

Midiendo respecto al osciloscopio se puede llegar a notar un pequeño desfase esto debido a las prioridades, pero es mínimo y esto sucede después de un tiempo.

4. Modifique las prioridades y analice las últimas dos preguntas nuevamente.

Al realizar los cambios e implementar al momento de encender los Leds se sigue sin poder apreciar el led que prende primero, a su vez midiendo respecto al osciloscopio ahora no alcanzamos a distinguir el desfase que se realizaba después de un tiempo esto al traslape de tareas.



2. Generación de tareas.

2.1. Utilizando FreeRTOS, genere 7 tareas:

1. La primera tarea enviará por puerto UART el mensaje: **Unidad Profesional Interdisciplinaria de Ingenieria Campus Zacatecas IPN**. Considere una velocidad de transmisión de 300 baudios (el más lento que permite el entorno Arduino IDE). Durante el envío considere que una salida digital esté en alto.
2. La segunda tarea enviará por puerto UART el mensaje: **1**. Considere una velocidad de transmisión de 300 baudios (el más lento que permite el entorno Arduino IDE). Durante el envío considere que una salida digital esté en alto.
3. La tercera tarea enviará por puerto UART el mensaje: **2**. Considere una velocidad de transmisión de 300 baudios (el más lento que permite el entorno Arduino IDE). Durante el envío considere que una salida digital esté en alto.
4. La cuarta tarea enviará por puerto UART el mensaje: **3**. Considere una velocidad de transmisión de 300 baudios (el más lento que permite el entorno Arduino IDE). Durante el envío considere que una salida digital esté en alto.
5. La quinta tarea enviará por puerto UART el mensaje: **4**. Considere una velocidad de transmisión de 300 baudios (el más lento que permite el entorno Arduino IDE). Durante el envío considere que una salida digital esté en alto.
6. La sexta tarea enviará por puerto UART el mensaje: **5**. Considere una velocidad de transmisión de 300 baudios (el más lento que permite el entorno Arduino IDE). Durante el envío considere que una salida digital esté en alto.
7. La séptima tarea enviará por puerto UART el mensaje: **6**. Considere una velocidad de transmisión de 300 baudios (el más lento que permite el entorno Arduino IDE). Durante el envío considere que una salida digital esté en alto.

2.2. Considere las mismas prioridades todas las tareas y analice lo siguiente:

1. ¿El mensaje por puerto UART tiene alguna consistencia u orden?

Pierde el orden por completo y se intercalan tareas entre sí.

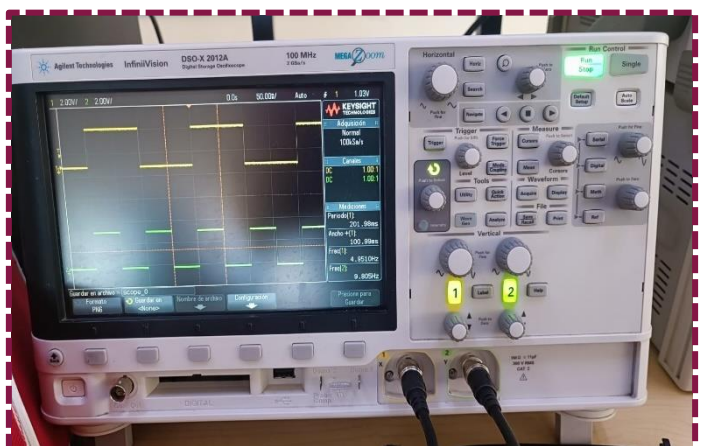
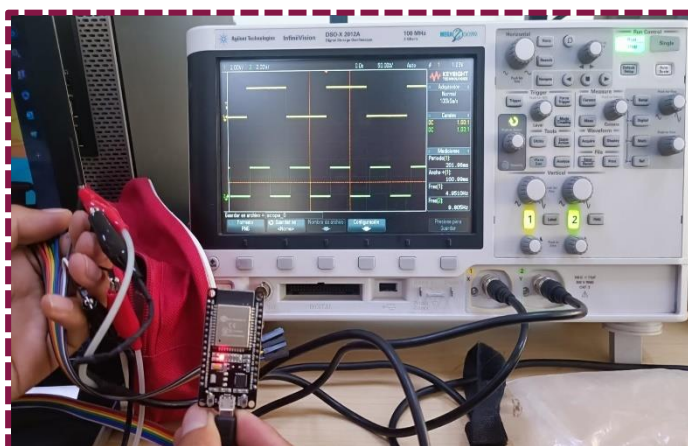
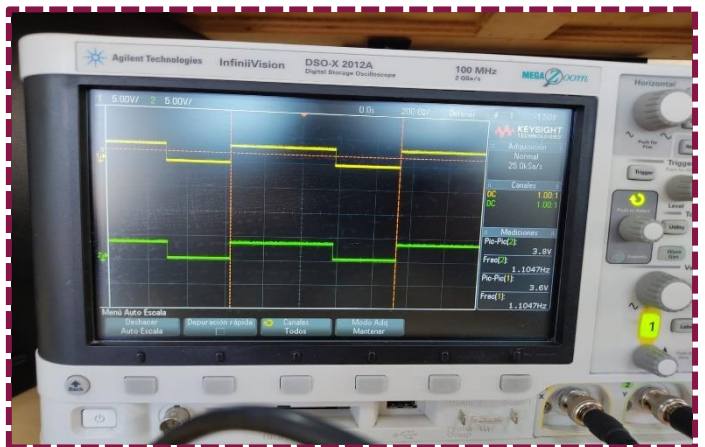
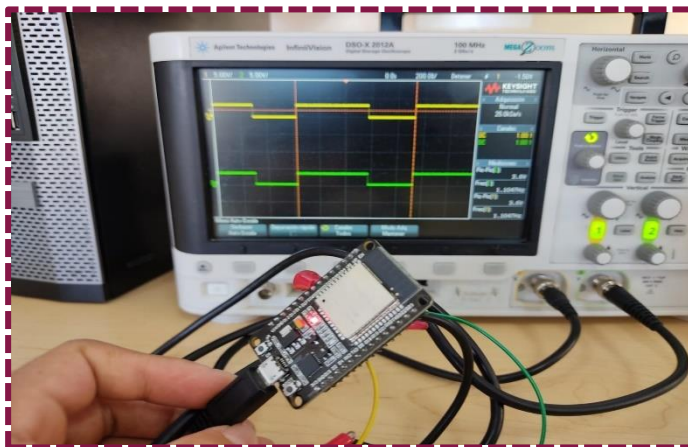

```

Unidad Profesional Interdisciplinaria de Ingenieria Campus Zacatecas IPN.
3.E (10367) uart: uart_write_bytes(1285): uart driver error
E (10368) uart: uart_get_buffered_data_len(1380): uart driver error
E (14635) uart: uart_get_buffered_data_len(1380): uart driver error
E (14637) uart: uart_write_bytes(1285): uart driver error
6.1.
2.
3.
4.
5.
Unidad Profesional Interdisciplinaria de Ingenieria Campus Zacatecas IPN.
6.
1.
2.
3.
4.
5.
Unidad Profesional Interdisciplinaria de Ingenieri

```

2. Utilizando el osciloscopio, tome como base la salida digital de la tarea 2. Compare esta salida con las tareas 3, 4, 5, 6 y 7. Analice los resultados.

Se puede apreciar cuando las salidas digitales se ponen en alto al momento de mandar los mensajes y al terminar se ponen en bajo, se puede distinguir diferencia en ciertos casos pero en otros se nota que se quieren activar a la vez por lo que se supone ahí es donde ocurre que los mensajes se corten entre ellos.



2.3. Considere al menos 4 cambios en las prioridades y analice lo siguiente:

1. ¿El mensaje por puerto UART tiene alguna consistencia u orden?

Ya empieza a notarse una secuencia, pero las tareas que no cambiaron de prioridad se llegan a intercambiar en ocasiones como se muestra en la imagen.

```
1.
6.5.
4.
3.
2.
1.
Unidad Profesional Interdisciplinaria de Ingenieria Campus Zacatecas IPN.
3.
2.
1.
6.5.
4.
3.
2.
1.
Unidad Profesional Interdisciplinaria de Ingenieria Campus Zacatecas IPN.
5.6.
4.
3.
2.
```

2. Para una de las 4 configuraciones utilizadas, tome como base la salida digital de la tarea 2. Compare esta salida con las tareas 3, 4, 5, 6 y 7. Analice los resultados.



3. Trabajo con los estados de las tareas.

3.1. Utilizando FreeRTOS, genere 3 tareas.

1. Tarea 1, Envío de mensaje por UART, mensaje y velocidad libre.
2. Tarea 2, Envío de mensaje por UART, mensaje y velocidad libre.
3. Tarea 3, Envío de mensaje por UART, mensaje y velocidad libre.

3.2. La tarea 1, se deberá ejecutar 1 sola vez y después será eliminada.

3.3. La tarea 2, se deberá suspender por 2 segundos y después habilitarse nuevamente.

3.4. La tarea 3, se ejecuta indefinidamente. Deberá tener máxima prioridad.

3.5. Reporte una captura de los mensajes obtenidos por consola.

```
#if CONFIG_FREERTOS_UNICROE
static const BaseType_t app_cpu = 0;
#else
static const BaseType_t app_cpu = 1;
#endif

const char msg1[]="Tarea 1: Yo soy la bomba ";
const char msg2[]="Tarea 2: I am the bombe ";
const char msg3[]="Tarea 3 HaAAAAA ";
static TaskHandle_t Tarea01 = NULL;
static TaskHandle_t Tarea02 = NULL;
static TaskHandle_t Tarea03 = NULL;

//Para la tarea 1
void Tarea1(void *parameter) {
int msg1_len = strlen(msg1);
while(1){
    Serial.println();
    for(int i=0;i<msg1_len;i++){
        Serial.print(msg1[i]);
    }
    Serial.println();
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
}

//Para la tarea 2
void Tarea2(void *parameter) {
int msg2_len = strlen(msg2);
while(1){
    Serial.print('-');
    for(int i=0;i<msg2_len;i++){
        Serial.print(msg2[i]);
    }
}
```

```

    }
    Serial.println();
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
}

//Para la tarea 3
void Tarea3(void *parameter) {
int msg3_len = strlen(msg3);
while(1){
    Serial.print('-');
    for(int i=0;i<msg3_len;i++){
        Serial.print(msg3[i]);
    }
    Serial.println();
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
}

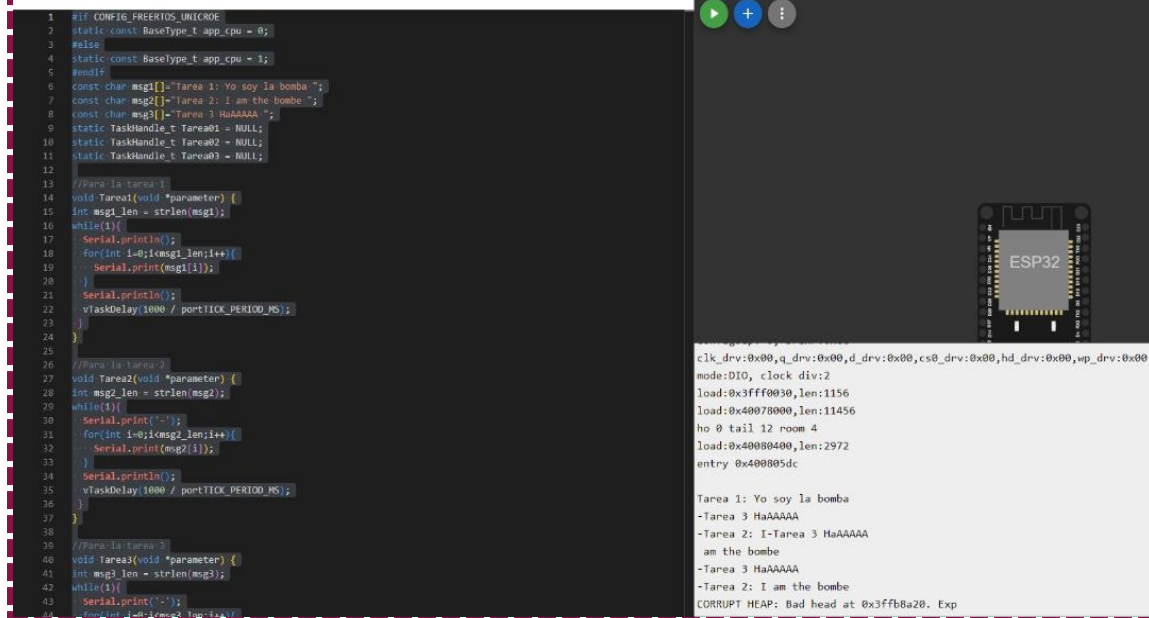
void setup(){
Serial.begin(300);
    xTaskCreatePinnedToCore(
        // Traer función
        Tarea1,
        "Tarea1",
        1024,
        NULL,
        2,
        &Tarea01,
        app_cpu);
    Serial.begin(300);
    xTaskCreatePinnedToCore(
        //Traer función
        Tarea2,
        "Tarea2",
        1024,
        NULL,
        1,
        &Tarea02,
        app_cpu);
    Serial.begin(300);
    xTaskCreatePinnedToCore(
        //Traer función
        Tarea3,
        "Tarea3",
        1024,
        NULL,
        7,
        &Tarea03,
        app_cpu);
}

```

```

void loop(){
  while(1){
    vTaskDelete(Tarea01);
    vTaskSuspend(Tarea02);
    vTaskDelay(2000 / portTICK_PERIOD_MS);
    vTaskResume(Tarea02);
    vTaskDelay(2000 / portTICK_PERIOD_MS);
  }
}

```



```

1 #if CONFIG_FREERTOS_UNICORE
2 static const BaseType_t app_cpu = 0;
3 #else
4 static const BaseType_t app_cpu = 1;
5 #endif
6 const char msg1[] = "Tarea 1: Yo soy la bomba.";
7 const char msg2[] = "Tarea 2: I am the bombe.";
8 const char msg3[] = "Tarea 3 HaAAAAA";
9 static TaskHandle_t Tarea01 = NULL;
10 static TaskHandle_t Tarea02 = NULL;
11 static TaskHandle_t Tarea03 = NULL;
12
13 //Para la tarea 1
14 void Tarea1(void *parameter) {
15   int msg1_len = strlen(msg1);
16   while(1){
17     Serial.println();
18     for(int i=0; i<msg1_len; i++){
19       Serial.print(msg1[i]);
20     }
21     Serial.println();
22     vTaskDelay(1000 / portTICK_PERIOD_MS);
23   }
24 }
25
26 //Para la tarea 2
27 void Tarea2(void *parameter) {
28   int msg2_len = strlen(msg2);
29   while(1){
30     Serial.print("");
31     for(int i=0; i<msg2_len; i++){
32       Serial.print(msg2[i]);
33     }
34     Serial.println();
35     vTaskDelay(1000 / portTICK_PERIOD_MS);
36   }
37 }
38
39 //Para la tarea 3
40 void Tarea3(void *parameter) {
41   int msg3_len = strlen(msg3);
42   while(1){
43     Serial.print("");
44     for(int i=0; i<msg3_len; i++){
45       Serial.print(msg3[i]);
46     }
47     Serial.println();
48     vTaskDelay(1000 / portTICK_PERIOD_MS);
49   }
50 }

```

clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
 mode:DIO, clock div:2
 load:0x3ffff030,len:1156
 load:0x40078000,len:11456
 ho 0 tail 12 room 4
 load:0x40080400,len:2972
 entry 0x400805dc

Tarea 1: Yo soy la bomba
 -Tarea 3 HaAAAAA
 -Tarea 2: I-Tarea 3 HaAAAAA
 am the bombe
 -Tarea 3 HaAAAAA
 -Tarea 2: I am the bombe
 CORRUPT HEAP: Bad head at 0x3ffb8a20. Exp

```

load:0x3ffff030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc

Tarea 1: Yo soy la bomba
-Tarea 3 HaAAAAA
-Tarea 2: I-Tarea 3 HaAAAAA
  am the bombe
-Tarea 3 HaAAAAA
-Tarea 2: I am the bombe
CORRUPT HEAP: Bad head at 0x3ffb8a20. Exp

```

CONCLUSIONES.

4. Dentro de las conclusiones considere lo siguiente:

1. Ventajas y desventajas de utilizar RTOS.

La principal ventaja de este tipo de sistemas es la velocidad, el determinismo y la consistencia. Nos permite gestionar de sistemas complejas con la realización de múltiples operaciones en paralelo. La priorización de tareas nos permite garantizar operaciones críticas. Además de que no consume mucho recurso de hardware como memorias. No obstante, este tipo de sistemas presentan algunas desventajas por ser más complejos en su diseño en general, la curva de aprendizaje requiere tiempo y esfuerzo. Y también tareas con programación básica, se complican un poco usando RTOS. Debemos de saber dónde aplicarlos y dónde no. Ese es el secreto.

2. En qué condiciones considera que se debería implementar un RTOS.

Se debe de aplicar RTOS realmente cuando sea requerido. Cuando la aplicación tiene requisitos estrictos de tiempo real y debe responder a eventos críticos en plazos determinados, como sistemas de control de automóviles, dispositivos médicos, sistemas de seguridad, etc.

Otra condición para aplicar RTOS es que se requiera gestión completa de tareas y la ejecución simultanea de múltiples procesos, cuando se requiera priorizar tareas, y cuando se espera que la aplicación escale o se adapte a diferentes plataformas para no tener que rediseñar el código por completo.

REPOSITORIO.

<https://github.com/DonBerraco/RTOS>

BIBLIOGRAFÍA.

FreeRTOS. (2023, 15 agosto). FreeRTOS - market leading RTOS (Real time operating system) for embedded systems with internet of things extensions. <https://www.freertos.org/>

Alfredo De La Barrera González + ; Diego Alejandro Garnica Arriaga + ; Pedro Guevara Lopez + Escuela Superior De Ingeniería Mecánica Y Eléctrica (Culhuacán) - IPN, México (2012). Comparativa de sistemas operativos de tiempo real y sistemas operativos de tiempo compartido, para plataforma intel x86 y sus compatibles: AlephZero-Comprendamos No.63 [en línea] (25/09/2023). <http://www.comprendamos.org/alephzero/63/comparativadesi.html>

N.C. Audsley , A. Burns , M. F. Richardson , A. J. Wellings: Hard Real-Time

Real-Time Operating Systems Book 1 - The Theory" y "Real-Time Operating Systems Book 2 - The Practice" por Jim Cooling:

Scheduling: The Deadline-Monotonic Approach. Proc. IEEE Workshop on RealTime Operating Systems and Software (1991).

A. Burns & A. Wellings: Real-Time Systems and Programming Languages. Addison Wesley, ISBN 90-201-40365-x