Fecha de Entrega:

TRABAJO PRÁCTICO:

Introducción a la programación

Competencias a desarrollar

- Aplicar estructuras de control (condicionales, ciclos, funciones) en C++/Python para resolver problemas técnicos.
- Analizar y modelar situaciones reales del entorno industrial mediante programación estructurada.
- Utilizar herramientas de desarrollo para programar, probar y depurar aplicaciones en C++/Python.

Objetivo general

Desarrollar en los estudiantes la capacidad de aplicar estructuras de programación en C++/Python para resolver problemas reales en el ámbito de la ingeniería mecatrónica, mediante el análisis de situaciones industriales, el diseño de soluciones algorítmicas, y la implementación estructurada de código funcional.

E1. Uso funciones estándar de FreeRTOS de xTaskCreate(), xTaskCreatePinnedToCore()

Crear cuatro tareas: El código debe definir cuatro tareas: taskFunction1, taskFunction2, taskFunction3 y taskFunction4.
Prioridades de tarea: Todas las tareas se crean con “PRIORITY”. Esto significa que tienen la misma prioridad mediante un define en cabecera.
Asignación de tareas a núcleos: taskFunction1 está asignada al núcleo 0 (CPU0), mientras que las otras tres tareas están asignadas al núcleo 1 (CPU1), en caso de disponer de un microcontrolador dual corre. ¿Qué se puede observar si usamos “tskNO_AFFINITY”?
Retrasos de tarea: taskFunction1 tiene un retraso de 1000 ms, mientras que las otras tres tienen un retraso de 500 ms.
Variar los diferentes parámetros como ser STACK de memoria, prioridades núcleo y observar las diferentes respuestas en las diferentes tareas.

E2. Colas en FreeRTOS (xQueue)

Crear una cola (**xQueue**), para esta cola simple, eje. tipo entero, char, etc, para intercambiar datos entre dos tareas. Una tarea envía enteros a la cola la otra tarea recibe esos enteros y los imprime en pantalla. Todo esto usando un solo núcleo, luego modificar para ambos núcleos. (**TareaProductora**) envía enteros a una cola (**xQueue**). (**TareaConsumidora**) recibe esos enteros e imprime los valores.

E3. Comunicación entre tareas con estructuras de datos usando (xQueue)

Usar una estructura de datos (por ejemplo, typedef *struct {int id;float valor;}* *SensorData;)* como cola de datos, ver la diferencia respecto al ejercicio E2.
Se pide para el ejercicio:
Cada mensaje que se pasa por la cola contendrá un id y un valor flotante, como podría suceder al tener múltiples sensores o fuentes de datos.
Definir una estructura "SensorData" con un id y un valor.
La tarea productora envía cada segundo un dato con id fijo y valor creciente.
La tarea consumidora recibe el dato desde la cola e imprime su contenido por consola

E4. Detección de eventos por interrupción con (xQueue)

Implementar una interrupción externa mediante un botón conectado a un GPIO.
La ISR envía un evento a través de una cola (xQueue) y una tarea receptora lo imprime por consola.
Cuando se detecta un flanco de bajada en el botón, se genera un evento que es enviado desde la ISR a una cola.
Una tarea en FreeRTOS espera datos en la cola y, al recibirlos, informa por consola que se ha producido un evento.
Utilidad: manejar eventos asincrónicos sin bloquear el sistema.

E5. Sincronización de tarea con interrupción usando semáforo binario

Detectar la pulsación de un botón mediante una interrupción en un GPIO y sincronizar la respuesta de una tarea usando un semáforo binario.
Al presionar un botón conectado al GPIO, la ISR activa un semáforo binario. Una tarea espera dicho semáforo y, al recibirlo, imprime un mensaje indicando que se detectó el evento.
Utilidad: Usado desde ISR

E6. Protección de sección crítica con Mutex en FreeRTOS

Implementar la exclusión mutua entre dos tareas que acceden concurrentemente a un recurso compartido (la consola) utilizando un mutex. Dos tareas intentan imprimir mensajes por consola. Para evitar condiciones de carrera y mezcla de mensajes, ambas tareas toman un mutex antes de imprimir y lo liberan después. Esto asegura que solo una tarea a la vez pueda acceder a la sección crítica. Además, simular una carga de trabajo con vTaskDelay() mientras se tiene el mutex.

Utilidad: para proteger cualquier recurso compartido:
UART
I2C/SPI