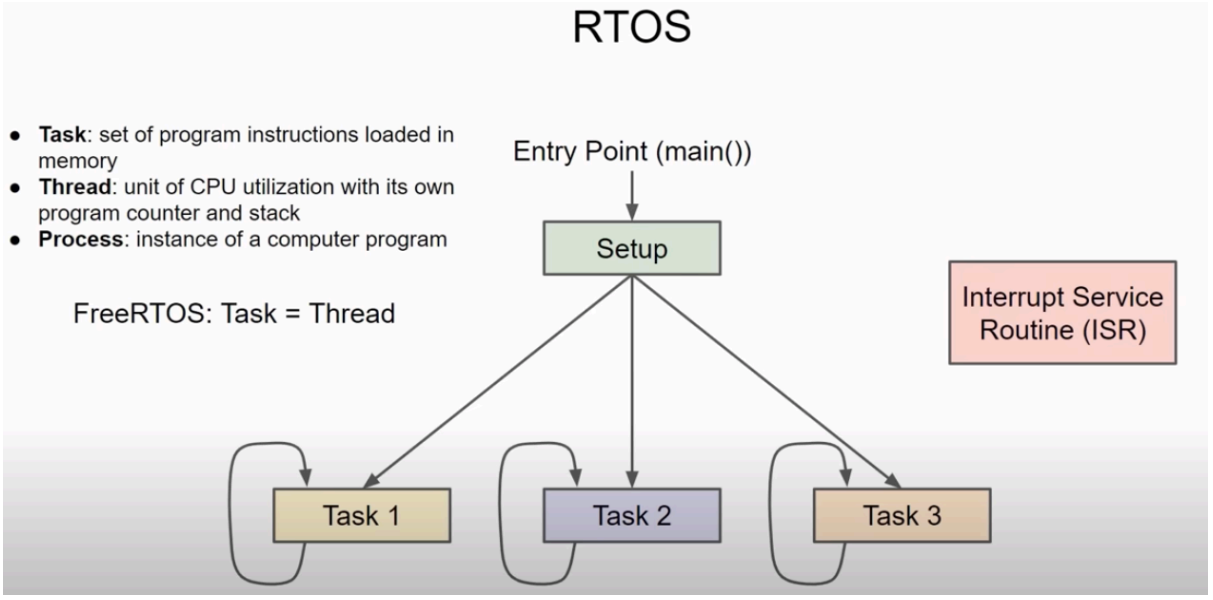
Vídeo 1: What is a Real-Time Operating System (RTOS)?

Cuestión 1: ¿Qué instrucciones se suelen separar entre el *setup()* y el *loop()* cuándo programamos en el Arduino IDE?

Respuesta: En el setup() se ponen las instrucciones que solo se tienen que ejecutar una vez, como la configuraciones de pines (ejemplo: pinMode(13, OUTPUT), define el pin 13 como salida), la asignación de valores iniciales (como poner que un está apagado al iniciar) y la inicialización de bibliotecas y periféricos. Mientras que en el loop se ponen las instrucciones que se repiten, como la lectura de los sensores (leer el valor del pin 13 por ejemplo), el control de los dispositivos (encender un led) y la comunicación entre otros.

Cuestión 2: En el contexto de FreeRTOS, ¿cómo se relaciona una tarea y un hilo?

Respuesta: Las tareas son la implementación específica de los hilos**.** Una tarea es un conjunto de instrucciones cargadas en la memoria (unidad básica de ejecución independiente), mientras que un hilo es una unidad de ejecución dentro de un proceso (instancia de un programa). Aunque a veces en este contexto sus significados son muy similares.



Vídeo 2: GettingStarted with FreeRTOS.

Cuestión 1: ¿Qué instrucción permite detener el código de manera NO bloqueante, funciona con FreeRTOS Vanilla y es útil en multiprogramación?

Respuesta: La función VTaskDelay(), que básicamente le dice al scheduler (programa que se encarga de planificar) que haga otras tareas hasta que el tiempo especificado (en ticks no microsegundo) se acabe.

Cuestión 2: Si Elena lanza a ejecutar dos tareas: A y B, siendo sus prioridades 0 y 1, respectivamente, ¿cuál se va ejecutar primero?

Respuesta: La B,cuanto más alto el número de la prioridad, más prioridad tiene, por lo que se ejecutará primero la de prioridad 1.

Vídeo 3: Task Scheduling

Cuestión 1: Si detengo un proceso con la instrucción *vTaskSuspend()*, ¿cómo puedo continuarla? ¿Y borrarla?

Respuesta: Se puede continuar con ese proceso usando la instrucción vTaskResume(), pero esto lo lleva al apartado de Ready, es decir no se ejecutará hasta el siguiente tick (en caso de que tenga mayor prioridad que las tareas existentes). Para borrarla se puede usar vTaskDelete(), en el video recalca la necesidad de comprobar antes que la tarea exista (mediante *if(task !=NULL)*)

Cuestión 2: Tenemos dos tareas, A está suspendida y B está bloqueada, ¿se encuentran en un mismo estado? ¿Cuál es la diferencia?

Respuesta: No, no se encuentran en el mismo estado, ninguna de las dos está ejecutando, pero A no se va a ejecutar a menos que se le ordene lo contrario (con vTaskResume() por ejemplo), mientras que al bloqueada puede ejecutarse en cualquier momento, está esperando a que ocurra algo (como un retardo de tiempo causado por vTaskDelay()).

Vídeo 6: Mutex

Cuestión 1: Enumera las cuatro formas que existen para proteger una sección crítica y en qué se diferencia cada uno.

Respuesta: Queues (almacenan datos de forma temporal), Locks(permiten que una tarea bloquee un recurso hasta que termine de usarlo, vamos que aunque use una variable global no permite que otras tareas accedan a este mientras lo está usando) , Mutex (MUTual EXclusion, garantiza que solo una tarea accede al recurso compartido en un momento dado, se diferencias de los locks porque no son tan generales y no tienen dueño, la tarea que loa adquirió debe liberarlo) y los semáforos, que son mecanismos de señalización que pueden ser binarios, como un Mutex (aunque este caso más en el sentido de la sincronización que en el de la exclusión) ,o de conteo.

Vídeo 9: Hardware Interrupts

Cuestión 1: ¿Cuántos timers podemos encontrar en un ESP32 que trabajen a nivel de Hardware?

Respuesta: 4

Cuestión 2: Para crear una función asociada a un timer, ¿qué etiqueta debe llevar esta función? Pista: esta etiqueta permite que la función trabaje con la memoria interna del ESP32.

Respuesta: IRAM_ATTR

Cuestión 3: En el caso de usar semáforos, ¿con qué tipo de instrucciones podemos permitir que un timer trabaje con uno de estos?

Respuesta: xSemaphoreGiveFromISR() o xSemaphoreTake(),

Vídeo 12: Multicore Systems:

Cuestión 1: ¿Cuántos cores tiene un ESP32?

Respuesta: 2, el core de protocolo y el de aplicación

Cuestión 2: ¿Cuáles actividades son las recomendadas para cada core? Pista: core 0 (protocol core) y core 1 (application core).

Respuesta: El core 0 se encarga de la comunicación, por eso se encarga de gestionar el Wi-Fi, control del Bluetooth, etc. Mientras que el core 1 se encarga de ejecutar la aplicación del usuario: procesamiento de sensores, algoritmos de control, tareas de interfaz, etc.

Cuestión 3: ¿Por qué motivos salta el TIMER WATCHDOG TASK? ¿Cuál core es el que lo tiene? ¿Está disponible en todos los cores? ¿Qué ocurre si salta? ¿Qué prevé?

Respuesta: Salta cada vez que no se llama al planificador (scheduler) cada pocos segundos. Lo tiene el core 0 y previene que las comunicaciones queden atrapadas en un bucle y por defecto no está en el corer 1. Si salta el TIMER WATCHDOG TASK se te reiniciará el procesador.

Cuestión 4: ¿Cuáles son las ventajas de “pinnear” una tarea en su core?

Respuesta: Sabes la localización de los interruptores, tienes más control sobre el core, menos errores de caché y es más determinista