

## freeRTOS con ESP32 Eliminar,suspender y resumir tarea

El ESP32 tiene dos núcleos de procesamiento de 32 bits, lo que significa que puede ejecutar dos tareas simultáneamente. Cada núcleo puede ejecutar su propio programa y acceder a su propia memoria. Además de los dos núcleos, el ESP32 también tiene una unidad de procesamiento de señal digital (DSP), que se utiliza para tareas que requieren un procesamiento intensivo de señales, como el procesamiento de audio o vídeo.



En FreeRTOS, las tareas se eliminan utilizando la función `vTaskDelete()`. La función toma un argumento que es el identificador de la tarea que se va a eliminar. Al llamar a esta función, la tarea se elimina inmediatamente del sistema operativo y su memoria se libera.

Para suspender una tarea se utiliza la función `vTaskSuspend()`. La función toma un argumento que es el identificador de la tarea que se va a suspender y para resumir la tarea se utiliza la función `vTaskResume()` de la misma manera que la anterior toma un argumento que es el identificador.

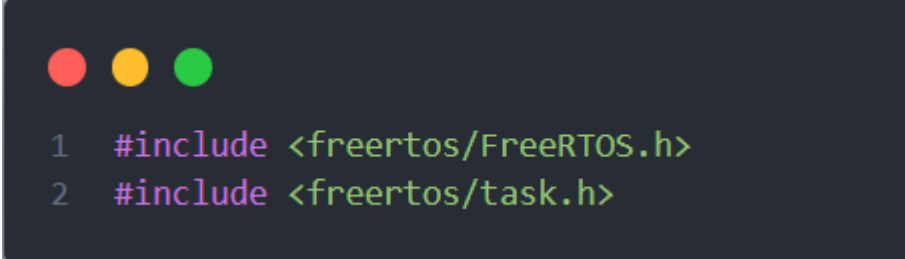
Es importante tener en cuenta que cuando se elimina una tarea, cualquier recurso que esté siendo utilizado por la tarea también se liberará. Por ejemplo, si una tarea está utilizando un semáforo o una cola, estos recursos se liberarán automáticamente cuando se elimina la tarea.

### Materiales necesarios:

- ESP32
- Cable USB
- Computadora con el IDE de Arduino instalado

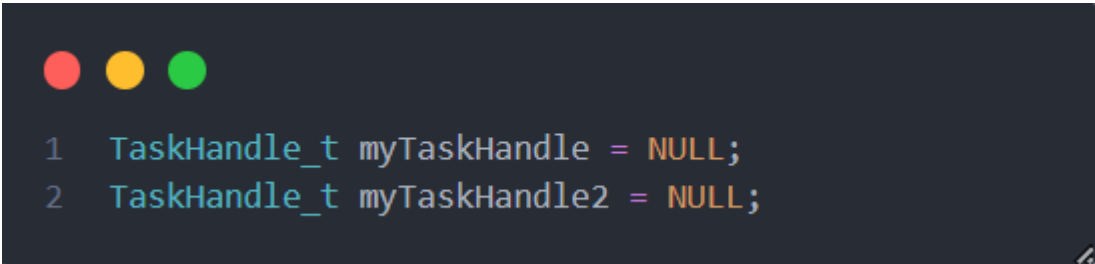
## Desarrollo

1. Para iniciar Incluimos las librerias en la parte superior del programa.



```
1  #include <freertos/FreeRTOS.h>
2  #include <freertos/task.h>
```

2. Se declaran dos variables de tipo TaskHandle\_t, que se utilizarán para almacenar los identificadores de las tareas creadas. El valor inicial de ambas variables es NULL, lo que significa que en ese momento no están asociadas a ninguna tarea en particular. Estas variables se utilizan para almacenar el manejador devuelto por la función que crea la tarea, lo que permite realizar operaciones sobre esa tarea en el futuro, como suspenderla, reanudarla o eliminarla.



```
1  TaskHandle_t myTaskHandle = NULL;
2  TaskHandle_t myTaskHandle2 = NULL;
```

3. Creamos una función llamada Demo\_Task con un contador que incrementa el valor de una variable local count en 1 en cada iteración del ciclo.
  - Imprime en la consola serial el mensaje "Demo\_Task printing.." utilizando la función Serial.println().
  - Hace una pausa de 1000 milisegundos utilizando la función vTaskDelay() de FreeRTOS. Esta función convierte el valor de tiempo dado en milisegundos a un valor de ticks necesario para hacer una pausa en la ejecución de la tarea.
  - Si la variable count es igual a 5, suspende la tarea identificada por myTaskHandle2 mediante la función vTaskSuspend().
  - Si la variable count es igual a 8, reanuda la tarea identificada por myTaskHandle2 mediante la función vTaskResume().
  - Si la variable count es igual a 10, elimina la tarea identificada por myTaskHandle2 mediante la función vTaskDelete().

```

1 void Demo_Task(void *arg)
2 {
3     int count = 0;
4     while(1){
5         count++;
6         Serial.println("Demo_Task printing..\n");
7         vTaskDelay(pdMS_TO_TICKS(1000));
8         if (count == 5)
9         {
10            vTaskSuspend(myTaskHandle2);
11            Serial.println("Demo_Task2 is suspended!\n");
12        }
13        if (count == 8)
14        {
15            vTaskResume(myTaskHandle2);
16            Serial.println("Demo_Task2 is resumed!\n");
17        }
18        if (count == 10)
19        {
20            vTaskDelete(myTaskHandle2);
21            Serial.println("Demo_Task2 is deleted!\n");
22        }
23    }
24 }
25

```

4. Crea una funcion llamada Demo\_Task2 con un contador que inicia en 0 y va sumando de 1 en 1 he imprime en cada interacción Demo\_Task2 printing.

```

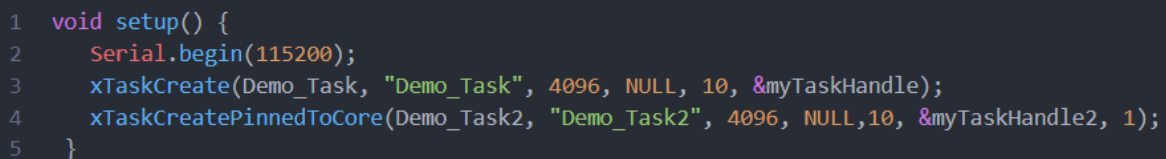
1 void Demo_Task2(void *arg)
2 {
3     for(int i=0;i<10;i++){
4         Serial.println("Demo_Task2 printing..\n");
5         vTaskDelay(pdMS_TO_TICKS(1000));
6     }
7 }

```

5. En el setup iniciamos el puerto serial posteriormente se crean dos tareas (también conocidas como procesos o hilos) utilizando la función "xTaskCreate" y "xTaskCreatePinnedToCore", que son funciones proporcionadas por FreeRTOS para crear tareas. Estas tareas se ejecutarán simultáneamente en diferentes núcleos del microcontrolador.

El primer parámetro en ambas funciones es el nombre de la función que se ejecutará como tarea (en este caso, "Demo\_Task" y "Demo\_Task2"). El segundo parámetro es el nombre que se le dará a la tarea. El tercer parámetro es el tamaño de la pila que se asignará a la tarea (4096 bytes en este caso). El cuarto parámetro es un puntero a cualquier dato que se desee pasar a la tarea (en este caso, NULL indica que no se pasará ningún dato). El quinto parámetro es la prioridad de la tarea. En este caso, ambas tareas tienen una prioridad de 10. Finalmente, el sexto parámetro es un puntero a una variable que contendrá un identificador para la tarea que se está creando.

Nota: xTaskCreatePinnedToCore es una función para crear una tarea como xTaskCreate con la diferencia que en esta podemos seleccionar el núcleo en el que queremos que se ejecute la tarea



```
1 void setup() {
2     Serial.begin(115200);
3     xTaskCreate(Demo_Task, "Demo_Task", 4096, NULL, 10, &myTaskHandle);
4     xTaskCreatePinnedToCore(Demo_Task2, "Demo_Task2", 4096, NULL, 10, &myTaskHandle2, 1);
5 }
```

Nota: El void loop lo dejamos vacío

6. Como último paso subimos el programa a nuestra ESP32 y se verá el monitor serial la ejecución de 2 tareas al mismo tiempo

Puedes encontrar el código completo y los pdf en la siguiente página de

GitHub: [https://github.com/GabrielCorUs/freeRTOS\\_ESP32/tree/main/TaskfreeRTTOS\\_ESP32](https://github.com/GabrielCorUs/freeRTOS_ESP32/tree/main/TaskfreeRTTOS_ESP32)