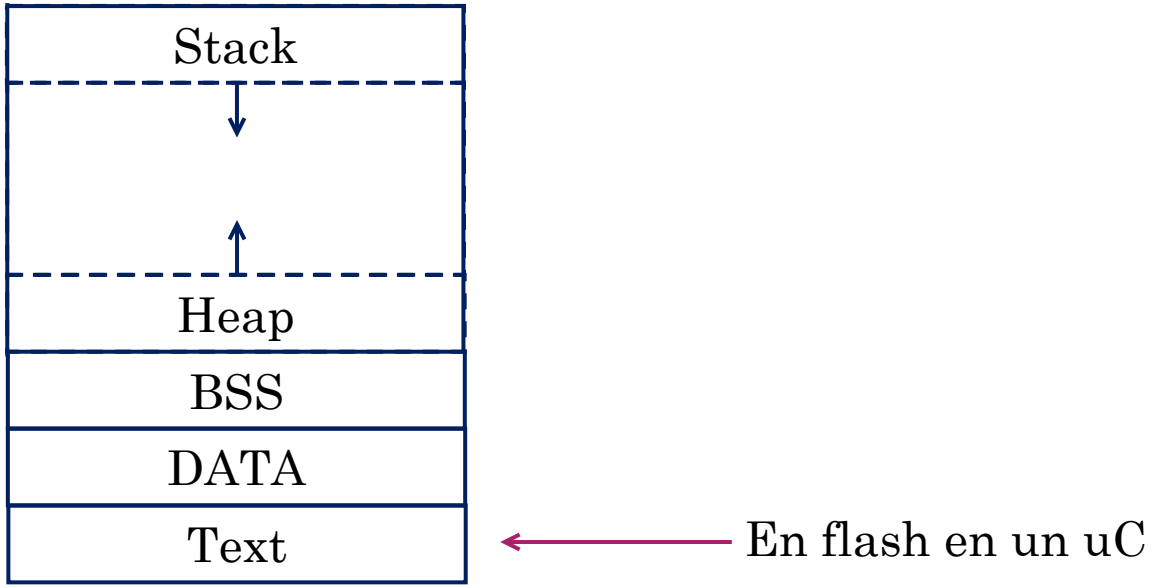
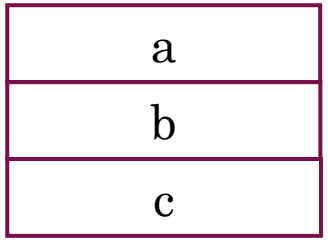

Stack
Heap
BSS
DATA
Text
En flash en un uC

```c
int imprime(int a, char * b)
{
    int c;
    c = cuadrado(a);
    printf("%d %s", c, b);
    return c;
}

int cuadrado(int d)
{
    int e;
    e = d * d;
    return e;
}
```

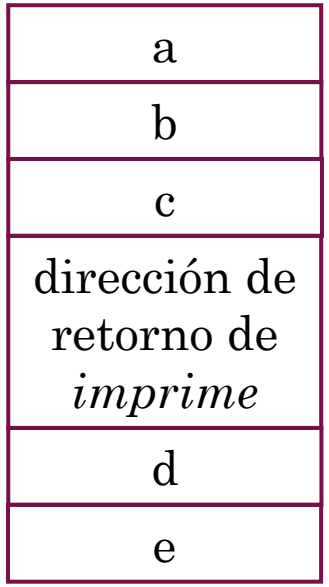
Justo antes de invocar *cuadrado*

```
int imprime(int a, char * b)
{
    int c;
    c = cuadrado(a);
    printf("%d %s", c, b);
    return c;
}

int cuadrado(int d)
{
    int e;
    e = d * d;
    return e;
}
```



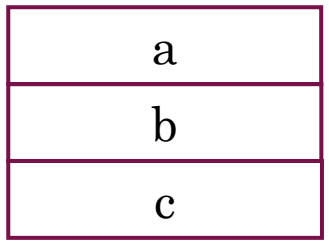
Justo después de invocar *cuadrado* y dentro de *cuadrado*

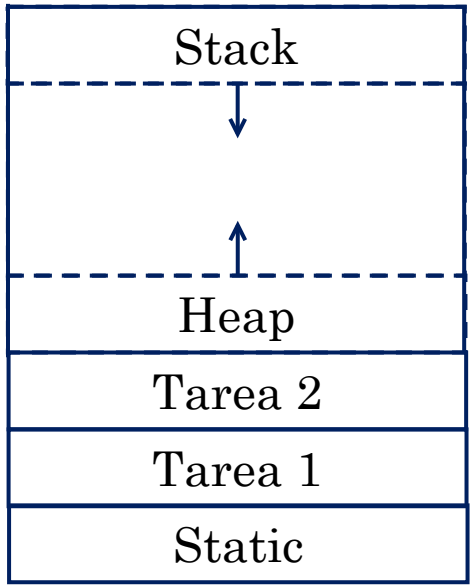
```c
int imprime(int a, char * b)
{
    int c;
    c = cuadrado(a);
    printf("%d %s", c, b);
    return c;
}

int cuadrado(int d)
{
    int e;
    e = d * d;
    return e;
}
```

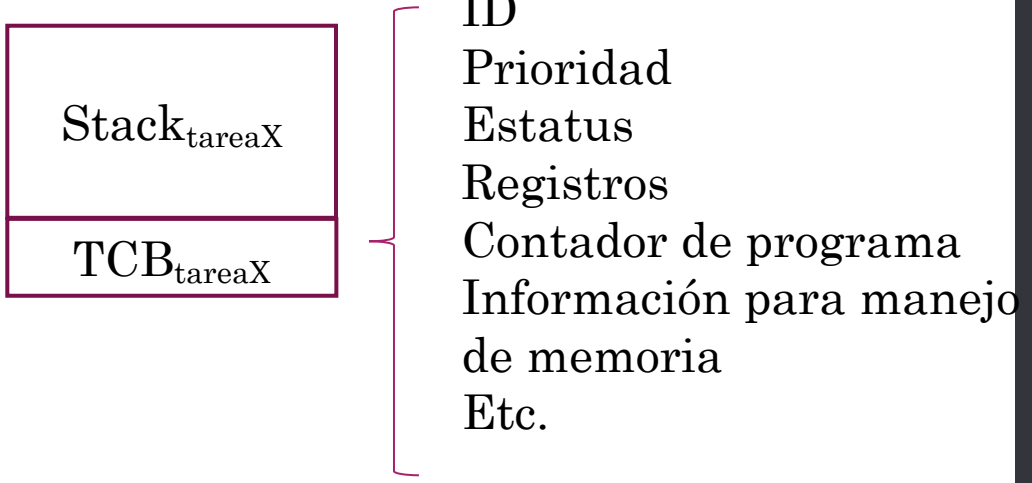


Cuando *cuadrado* ejecuta return

```
int imprime(int a, char * b)
{
    int c;
    c = cuadrado(a);
    printf("%d %s", c, b);
    return c;
}

int cuadrado(int d)
{
    int e;
    e = d * d;
    return e;
}
```



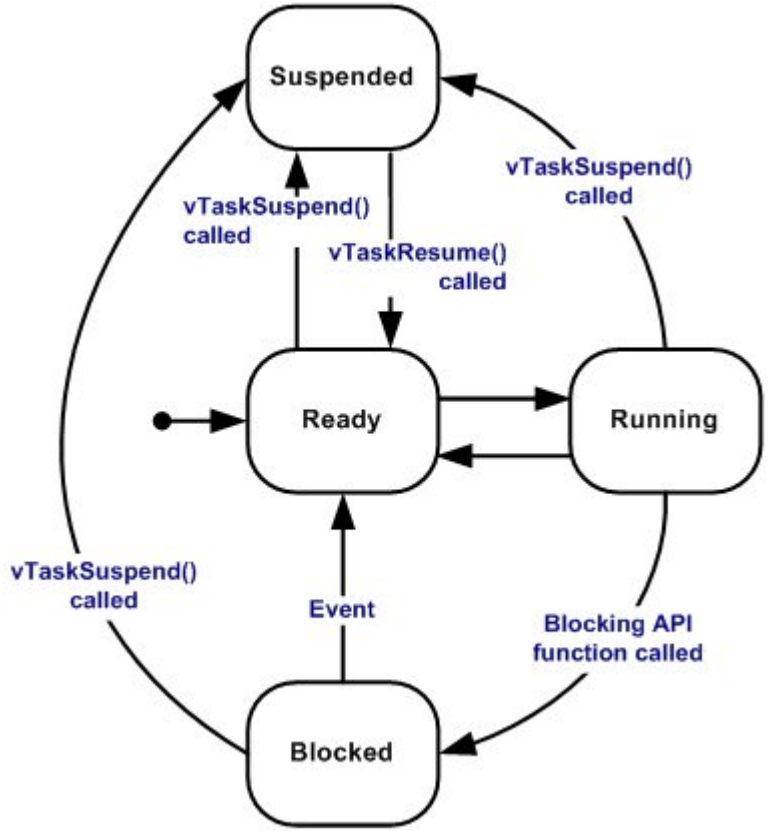

Stack
Heap
Tarea 2
Tarea 1
Static


Stack tareaX
TCB tareaX
ID
Prioridad
Estatus
Registros
Contador de programa
Información para manejo de memoria
Etc.

Estados de las tareas

- Ejecutándose.
- Lista.
- Bloqueada.
- Suspendida.
- Eliminada.

Estados de las tareas



https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/freertos_idf.html

```c
#include <stdio.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include "esp_timer.h"
```

```c
TaskHandle_t handleTarea1 = NULL;
TaskHandle_t handleTarea2 = NULL;
```

```c
void app_main(void)
{
  xTaskCreate(tarea1, "tarea1", 2048, NULL, 10, &handleTarea1);
  xTaskCreate(tarea2, "tarea2", 2048, NULL, 10, &handleTarea2);
}
```

```c
void tarea1(void *arg)
{
  int cnt = 0;
  while (1)
  {
    printf("Tarea 1 corriendo. Cnt: %d. Tiempo: %lld\n", cnt++, esp_timer_get_time());
    control_tareas(cnt, &handleTarea2);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

void tarea2(void *arg)
{
  int cnt = 0;
  while (1)
  {
    printf("Tarea 2 corriendo. Cnt: %d. Tiempo: %lld\n", cnt++, esp_timer_get_time());
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}
```

```c
void control_tareas(int estado, TaskHandle_t *taskHandle)
{
  if (*taskHandle == NULL)
  {
    return;
  }

  const char *taskName = pcTaskGetName(*taskHandle); /* Obtener el nombre de la tarea */

  switch (estado)
  {
    case 5:
    {
      printf("Tarea '%s' suspendida\n", taskName);
      vTaskSuspend(*taskHandle);
      break;
    }
    case 10:
    {
      printf("Tarea '%s' resumida\n", taskName);
      vTaskResume(*taskHandle);
      break;
    }
    case 15:
    {
      printf("Tarea '%s' eliminada\n", taskName);
      vTaskDelete(*taskHandle);
      *taskHandle = NULL;
      break;
    }
    default:
      break;
  }
}
```

```
Tarea 1 corriendo. Cnt: 0. Tiempo: 308084
Tarea 2 corriendo. Cnt: 0. Tiempo: 309065
Tarea 1 corriendo. Cnt: 1. Tiempo: 1307048
Tarea 2 corriendo. Cnt: 1. Tiempo: 1307130
Tarea 1 corriendo. Cnt: 2. Tiempo: 2307048
Tarea 2 corriendo. Cnt: 2. Tiempo: 2307130
Tarea 1 corriendo. Cnt: 3. Tiempo: 3307048
Tarea 2 corriendo. Cnt: 3. Tiempo: 3307130
Tarea 1 corriendo. Cnt: 4. Tiempo: 4307048
Tarea 'tarea2' suspendida
Tarea 1 corriendo. Cnt: 5. Tiempo: 5307047
Tarea 1 corriendo. Cnt: 6. Tiempo: 6307047
Tarea 1 corriendo. Cnt: 7. Tiempo: 7307047
Tarea 1 corriendo. Cnt: 8. Tiempo: 8307047
Tarea 1 corriendo. Cnt: 9. Tiempo: 9307047
Tarea 'tarea2' resumida
Tarea 2 corriendo. Cnt: 4. Tiempo: 4307130
Tarea 2 corriendo. Cnt: 5. Tiempo: 10307048
Tarea 1 corriendo. Cnt: 10. Tiempo: 10307130
Tarea 2 corriendo. Cnt: 6. Tiempo: 11307048
Tarea 1 corriendo. Cnt: 11. Tiempo: 11307130
Tarea 2 corriendo. Cnt: 7. Tiempo: 12307048
Tarea 1 corriendo. Cnt: 12. Tiempo: 12307130
Tarea 2 corriendo. Cnt: 8. Tiempo: 13307048
Tarea 1 corriendo. Cnt: 13. Tiempo: 13307130
Tarea 2 corriendo. Cnt: 9. Tiempo: 14307048
Tarea 1 corriendo. Cnt: 14. Tiempo: 14307130
Tarea 'tarea2' eliminada
Tarea 1 corriendo. Cnt: 15. Tiempo: 15307047
Tarea 1 corriendo. Cnt: 16. Tiempo: 16307047
Tarea 1 corriendo. Cnt: 17. Tiempo: 17307047
Tarea 1 corriendo. Cnt: 18. Tiempo: 18307047
```

vTaskSuspend(**NULL**);

Pasar un identificador NULL provocará que se suspenda la tarea que realiza la llamada.

La tarea no se ejecutará durante ese período, a menos que otra tarea llame a vTaskResume(xHandle).

vTaskDelete(**NULL**);

Autoeliminación de la tarea.

Sincronización de tareas

Event Groups

https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/freertos_idf.html?highlight=xeventgroupwaitbits#event-group-api

Event Groups

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_timer.h"
#include "freertos/event_groups.h"
```

```c
EventGroupHandle_t event_group;

const int sensors_ready = BIT0;
const int motor_ready = BIT1;
```

```c
void app_main(void)
{
  printf("configUSE_16_BIT_TICKS %d\n",configUSE_16_BIT_TICKS);
  event_group = xEventGroupCreate();
  xTaskCreate(control_sensores, "sensores", 2048, NULL, 10, NULL);
  xTaskCreate(control_motor, "motor", 2048, NULL, 10, NULL);
  xTaskCreate(control_ui, "ui", 2048, NULL, 10, NULL);
}
```

configUSE_16_BIT_TICKS

Aunque los grupos de eventos no están relacionados con los ticks, por razones de implementación interna, la cantidad de bits disponibles para usar en un grupo de eventos depende de la configuración configUSE_16_BIT_TICKS en FreeRTOSConfig.h. Si **configUSE_16_BIT_TICKS es 1**, cada grupo de eventos contiene **8 bits** utilizables (bit 0 a bit 7). Si **configUSE_16_BIT_TICKS es 0**, cada grupo de eventos tiene **24 bits** utilizables (bit 0 a bit 23).

```c
void control_sensores(void *params)
{
  int cnt = 0;
  while (true)
  {
    xEventGroupSetBits(event_group, sensors_ready);
    printf("Sensores leidos. Cnt: %d. Tiempo: %lld\n", cnt++, esp_timer_get_time());
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

void control_motor(void *params)
{
  int cnt = 0;
  while (true)
  {
    xEventGroupSetBits(event_group, motor_ready);
    printf("Estado del motor actualizado. Cnt: %d. Tiempo: %lld\n", cnt++, esp_timer_get_time());
    vTaskDelay(2000 / portTICK_PERIOD_MS);
  }
}

void control_ui(void *params)
{
  int cnt = 0;
  while (true)
  {
    xEventGroupWaitBits(event_group, sensors_ready | motor_ready, true, true, portMAX_DELAY);
    printf("****UI actualizada. Cnt: %d. Tiempo: %lld****\n", cnt++, esp_timer_get_time());
  }
}
```

```
EventBits_t xEventGroupWaitBits(
EventGroupHandle_t xEventGroup, /* El Event Group en el que se están probando los bits */
const EventBits_t uxBitsToWaitFor, /* Bits que se probarán dentro del grupo de eventos */
const BaseType_t xClearOnExit,     /* Limpiar los bits al retornar */
const BaseType_t xWaitForAllBits, /* Esperar todos los bits */
TickType_t xTicksToWait             /* Tiempo de espera*/
)
```

Si **xClearOnExit** se establece en TRUE, entonces todos los bits dentro de uxBitsToWaitFor se borrarán antes de que xEventGroupWaitBits() regrese si se cumplió la condición de espera (si la función regresa por un motivo distinto a un tiempo de espera). Si xClearOnExit se establece en FALSE, entonces los bits establecidos en el grupo de eventos no se modifican cuando la llamada a xEventGroupWaitBits() regresa.

Si **xWaitForAllBits** se establece en TRUE, xEventGroupWaitBits() regresará cuando se establezcan todos los bits en uxBitsToWaitFor o expire el tiempo de bloqueo especificado. Si xWaitForAllBits se establece en FALSE, xEventGroupWaitBits() regresará cuando se establezca cualquiera de los bits indicados en uxBitsToWaitFor o expire el tiempo de bloqueo especificado.

xTicksToWait

La cantidad máxima de tiempo (especificada en 'ticks') que se debe esperar hasta que se configuren los bits. Se puede usar el valor de **portMAX_DELAY** para bloquear indefinidamente (siempre que INCLUDE_vTaskSuspend esté configurada en 1 en FreeRTOSConfig.h).

Retorno:

El valor del grupo de eventos en el momento en que se establecieron los bits que se estaban esperando o en que expiró el tiempo de bloqueo.

Pruebe el valor de retorno para saber qué bits se establecieron. Si xEventGroupWaitBits() regresó porque expiró su tiempo de espera, entonces no se establecerán todos los bits que se estaban esperando. Si xEventGroupWaitBits() regresó porque se establecieron los bits que estaba esperando, entonces el valor devuelto es el valor del grupo de eventos antes de que se borraran automáticamente los bits en el caso de que el parámetro xClearOnExit se estableciera en TRUE.

Tarea:
Verifique usted lo que indica la documentación.

```
configUSE_16_BIT_TICKS 0
Sensores leidos. Cnt: 0. Tiempo: 309546
Estado del motor actualizado. Cnt: 0. Tiempo: 310493
****UI actualizada. Cnt: 0. Tiempo: 311541****
Sensores leidos. Cnt: 1. Tiempo: 1308026
Estado del motor actualizado. Cnt: 1. Tiempo: 2308025
Sensores leidos. Cnt: 2. Tiempo: 2308043
****UI actualizada. Cnt: 1. Tiempo: 2308106****
Sensores leidos. Cnt: 3. Tiempo: 3308025
Estado del motor actualizado. Cnt: 2. Tiempo: 4308025
Sensores leidos. Cnt: 4. Tiempo: 4308043
****UI actualizada. Cnt: 2. Tiempo: 4308106****
Sensores leidos. Cnt: 5. Tiempo: 5308025
Estado del motor actualizado. Cnt: 3. Tiempo: 6308025
Sensores leidos. Cnt: 6. Tiempo: 6308043
****UI actualizada. Cnt: 3. Tiempo: 6308106****
```

Queue

```
#include <stdio.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include "freertos/queue.h"
```

```
#define NUM_EJES   3

TaskHandle_t tarea1Handle = NULL;
TaskHandle_t tarea2Handle = NULL;
QueueHandle_t queue;
```

```
void app_main(void)
{
    queue = xQueueCreate(10, sizeof(uint8_t*));
    if (queue == 0)
    {
        printf("Error al crear la cola\n");
        while (1);
    }

   xTaskCreate(tarea1, "tarea1", 2048, NULL, 10, &tarea1Handle);
   xTaskCreate(tarea2, "tarea2", 2048, NULL, 10, &tarea2Handle);
 }
```

```c
void tarea1(void *arg)
{
    uint8_t buffer[NUM_EJES];
    uint8_t *pBuffer;
    static uint8_t val = 0;

    pBuffer = buffer;

    while (1)
    {
        for (uint8_t i = 0; i < NUM_EJES; i++)
        {
            buffer[i] = val++;
        }

        xQueueSend(queue, ( void * ) &pBuffer, (TickType_t)0);
        printf("Sensor leido\n");
        vTaskDelay(1000/ portTICK_PERIOD_MS);
    }

}

void tarea2(void *arg)
{
    uint8_t * ptr;

    while(1)
    {
        if( xQueueReceive(queue, &ptr, (TickType_t)5))
        {
            printf("Medicion del sensor recibida:\n");
            for (uint8_t i = 0; i < NUM_EJES; i++)
            {
                printf("[%d] = %d\n", i, ptr[i]);
            }
            vTaskDelay(1000/ portTICK_PERIOD_MS);
        }
    }
}
```

```
xQueueSend(
xQueue,              /* Cola en la cual se va a agregar el elemento */
pvItemToQueue, /* Elemento a agregar */
xTicksToWait      /* Espera máxima por espacio disponible */)
```

```
BaseType_t xQueueReceive(
QueueHandle_t xQueue,  /* Cola de la cual se va a tomar un elemento */
void *const pvBuffer,       /* Buffer donde se va a almacenar el elemento */
TickType_t xTicksToWait  /* Espera máxima para recibir un elemento */)
```

```
Sensor leido
Medicion del sensor recibida:
[0] = 0
[1] = 1
[2] = 2
Sensor leido
Medicion del sensor recibida:
[0] = 3
[1] = 4
[2] = 5
Sensor leido
Medicion del sensor recibida:
[0] = 6
[1] = 7
[2] = 8
```

Tarea:
Leer la documentación del ESP-IDF sobre los conceptos e instrucciones vistos en clase de GPIOs y tareas.