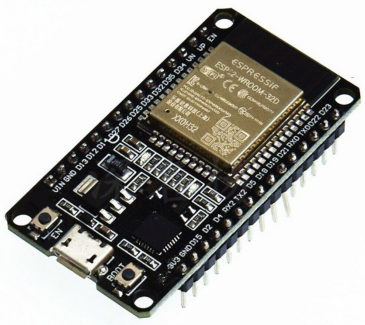
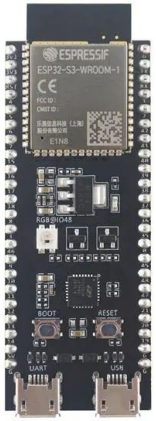
ESP32 y FreeRTOS

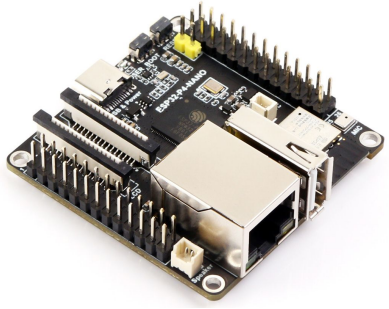
¿Qué es ESP32?
¿Qué es FreeRTOS?
Multiprocesamiento simétrico en ESP32
Modelización de procesos
Planificación de procesos
Recomendaciones
¿Cómo decide el planificador?

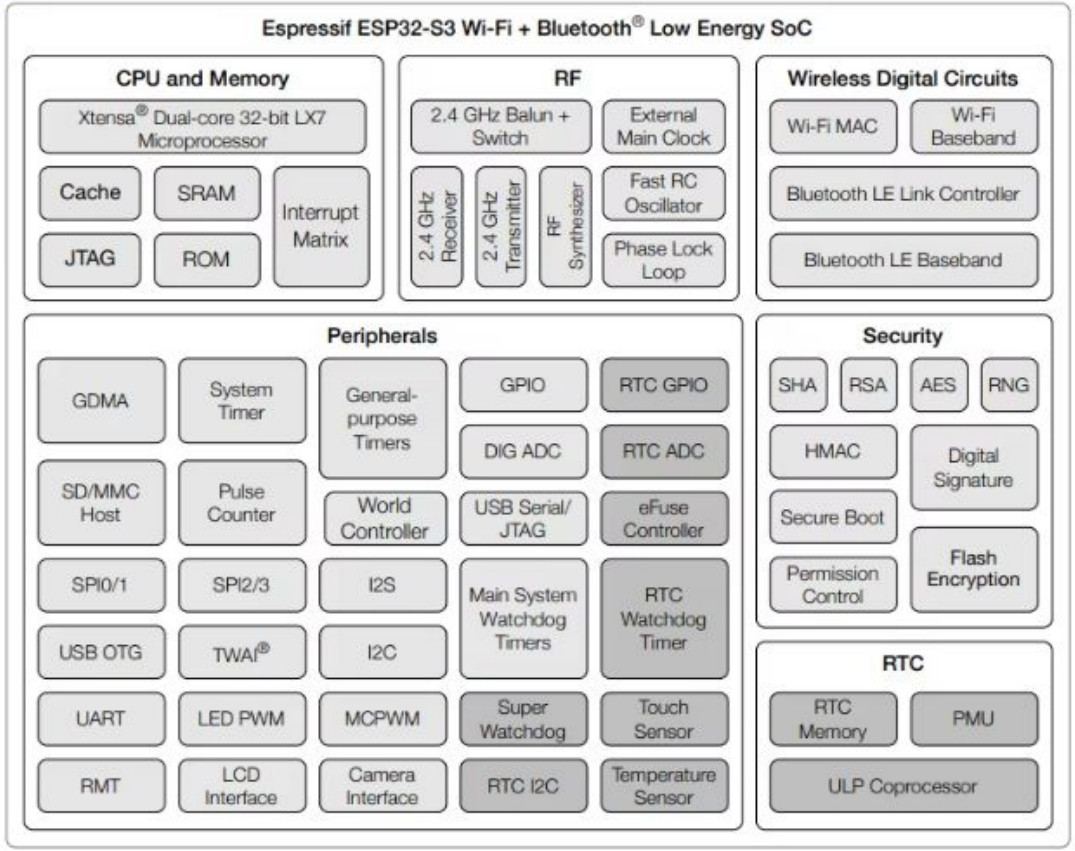
¿Qué es ESP32?






ESP-WROOM-32
211-161007
FCC ID:2AC7Z-ESPWROOM32




Espressif ESP32-S3 Wi-Fi + Bluetooth® Low Energy SoC
CPU and Memory
Xtensa® Dual-core 32-bit LX7 Microprocessor
Cache
SRAM
JTAG
ROM
Interrupt Matrix
RF
2.4 GHz Balun + Switch
External Main Clock
2.4 GHz Receiver
2.4 GHz Transmitter
RF Synthesizer
Fast RC Oscillator
Phase Lock Loop
Wireless Digital Circuits
Wi-Fi MAC
Wi-Fi Baseband
Bluetooth LE Link Controller
Bluetooth LE Baseband
Peripherals
GDMA
System Timer
General-purpose Timers
GPIO
RTC GPIO
DIG ADC
RTC ADC
SD/MMC Host
Pulse Counter
World Controller
USB Serial/ JTAG
eFuse Controller
SPI0/1
SPI2/3
I2S
Main System Watchdog Timers
RTC Watchdog Timer
USB OTG
TWAI®
I2C
UART
LED PWM
MCPWM
Super Watchdog
Touch Sensor
RMT
LCD Interface
Camera Interface
RTC I2C
Temperature Sensor
Security
SHA
RSA
AES
RNG
HMAC
Digital Signature
Secure Boot
Flash Encryption
Permission Control
RTC
RTC Memory
PMU
ULP Coprocessor

¿Qué es FreeRTOS?

- FreeRTOS (sistema operativo de tiempo real)

  -Sistema operativo para microcontroladores.

  -Tareas concurrentes → planificador por prioridades

- Sistema operativo de tiempo real (RTOS)

  -Garantiza respuestas predecibles en plazos definidos.

  -Prioriza el determinismo temporal para aplicaciones.

  -Responder a eventos externos con latencia mínima.

- Implementación FreeRTOS para ESP32 (25Kb)

  - (Task.c - Timer.c - Queue.c - Port.c - List.c)

Multiprocesamiento simétrico (SMP) en ESP32

- **Núcleos homogéneos:** Garantizan comportamiento consistente de código binario independientemente del núcleo de ejecución.
- **Arquitectura de memoria simétrica:**
  - Memoria accesible por ambos núcleos.
  - Hardware se encarga de acceso → bus de memoria
- **Sistemas de interrupciones cruzadas** (inter-processor interrups)
  - Interrupciones entre núcleos.

Núcleo 0

1. Mantenimiento del tiempo global → tick
2. Gestión de despertar
3. Planificación global → tareas listas
4. Hooks de usuario → función en cada tick

Núcleo 1

1.2. 3. NO
4. Hooks de usuario
5. Time slicing local → rotar tareas

Modelización de procesos TCB (Task Control Block)

tasks: Unidades concurrentes dentro de un único espacio de memoria

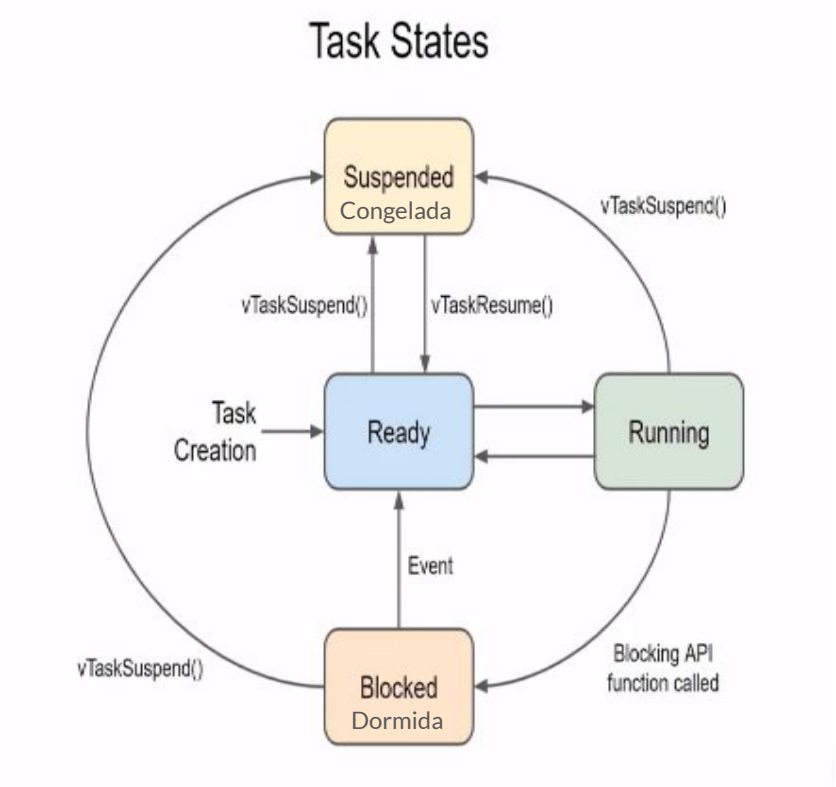
Espacio de memoria compartida: Todas las tareas comparten el mismo espacio de memoria. Esto significa que una tarea puede acceder directamente a la memoria de otra tarea, lo que facilita la comunicación, pero requiere mecanismos de sincronización adecuados.

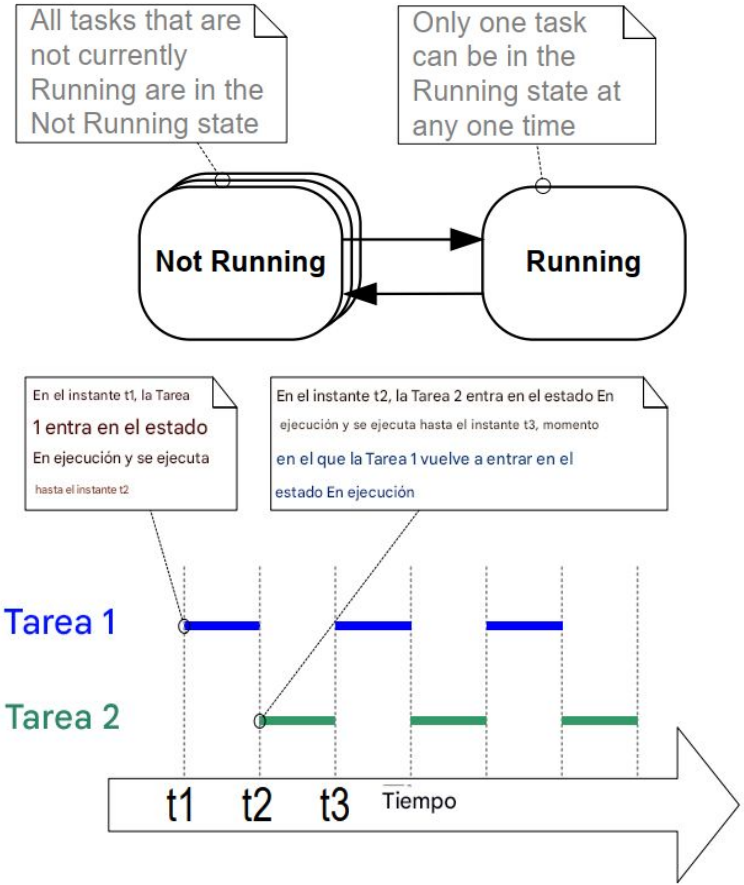
Cambio de contexto rápido: Al compartir espacio de memoria, el cambio de contexto entre tareas es más eficiente. Solo se necesita:

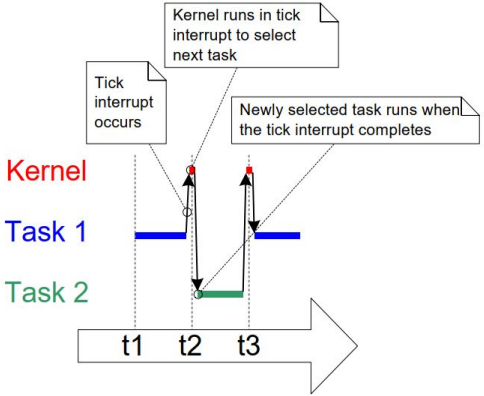
- Guardar registros del procesador
- Actualizar el puntero al TCB de la tarea
- Actualizar algunas estructuras internas

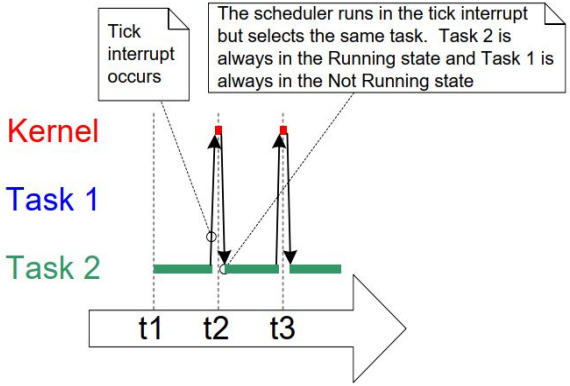
```
typedef struct tskTaskControlBlock
{
  volatile StackType_t *pxTopOfStack;   /* Puntero a la última posición de la pila donde se guardó el contexto */

/* Elementos de gestión de listas */
  ListItem_t xStateListItem;          /* Usado para posicionar la tarea en listas de estado del sistema */
  ListItem_t xEventListItem;          /* Usado para posicionar la tarea en listas de eventos */
  UBaseType_t uxPriority;             /* Prioridad de la tarea (0 a configMAX_PRIORITIES-1) *

/* Gestión de memoria */
  StackType_t *pxStack;               /* Puntero al inicio de la pila asignada */
  StackType_t *pxEndOfStack;     /* Puntero al final de la pila para verificar desbordamiento */

  char pcTaskName[configMAX_TASK_NAME_LEN]; /* Nombre descriptivo de la tarea*/

/* Estadísticas y depuración */
#if (configGENERATE_RUN_TIME_STATS == 1)
        uint32_t ulRunTimeCounter; /* Contador de tiempo de ejecución para estadísticas */
 #endif

#if configUSE_TASK_NOTIFICATIONS
        volatile uint32_t ulNotifiedValue[configTASK_NOTIFICATION_ARRAY_ENTRIES];
        volatile uint8_t ucNotifyState[configTASK_NOTIFICATION_ARRAY_ENTRIES];
#endif

#if (configUSE_TRACE_FACILITY == 1)
        UBaseType_t uxTCBNumber; /* Número único de identificación de la tarea */
        UBaseType_t uxTaskNumber; /* Número incremental asignado a cada tarea */
 #endif

#if (configUSE_MUTEXES == 1)
        UBaseType_t uxBasePriority; /* Prioridad base (para herencia de prioridades) */
        UBaseType_t uxMutexesHeld; /* Contador de mutexes adquiridos */
#endif

#if configUSE_CORE_AFFINITY
         UBaseType_t uxCoreAffinityMask; /* Máscara de afinidad para asignación a núcleos específicos */
 #endif

volatile BaseType_t xSchedulerState; /* Estado del planificador al momento de crear la tarea */

/* ... otros campos ... */
} tskTCB;
```

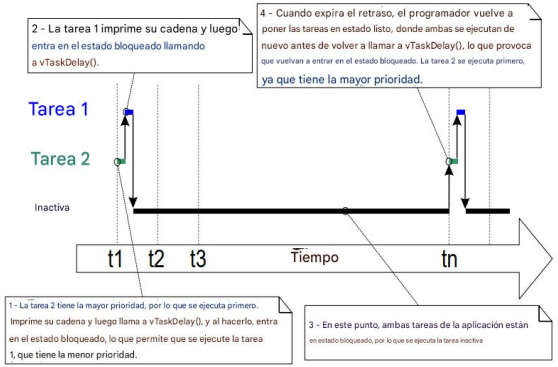
Planificación de procesos

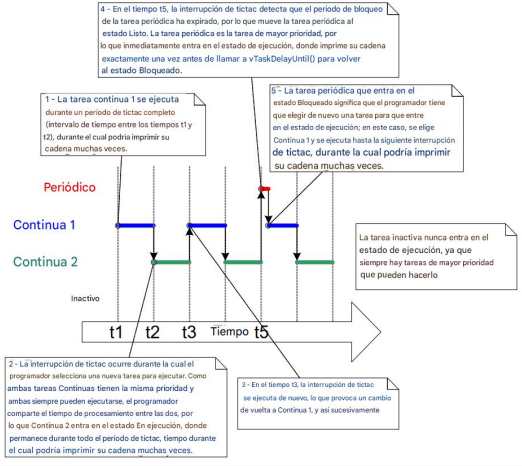


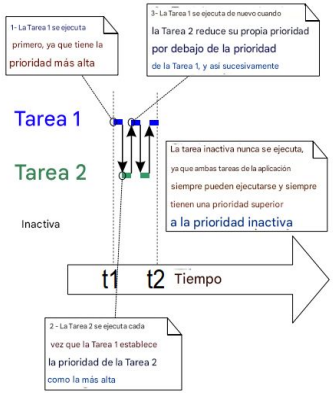


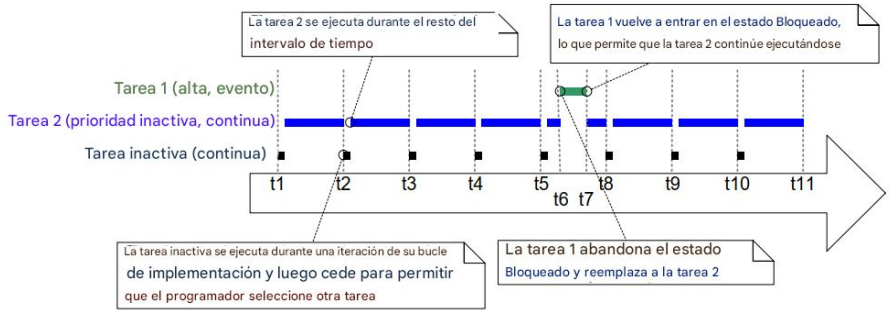

Kernel runs in tick interrupt to select next task
Tick interrupt occurs
Newly selected task runs when the tick interrupt completes
Kernel
Task 1
Task 2
t1
t2
t3


Tick interrupt occurs
The scheduler runs in the tick interrupt but selects the same task. Task 2 is always in the Running state and Task 1 is always in the Not Running state
Kernel
Task 1
Task 2
t1
t2
t3


2 - La tarea 1 imprime su cadena y luego entra en el estado bloqueado llamando a vTaskDelay().
4 - Cuando expira el retraso, el programador vuelve a poner las tareas en estado listo, donde ambas se ejecutan de nuevo antes de volver a llamar a vTaskDelay(), lo que provoca que vuelvan a entrar en el estado bloqueado. La tarea 2 se ejecuta primero, ya que tiene la mayor prioridad.
Tarea 1
Tarea 2
Inactiva
t1
t2
t3
Tiempo
tn
1 - La tarea 2 tiene la mayor prioridad, por lo que se ejecuta primero. Imprime su cadena y luego llama a vTaskDelay(), y al hacerlo, entra en el estado bloqueado, lo que permite que se ejecute la tarea 1, que tiene la menor prioridad.
3 - En este punto, ambas tareas de la aplicación están en estado bloqueado, por lo que se ejecuta la tarea inactiva


4 - En el tiempo t5, la interrupción de tictac detecta que el periodo de bloqueo de la tarea periódica ha expirado, por lo que mueve la tarea periódica al estado Listo. La tarea periódica es la tarea de mayor prioridad, por lo que inmediatamente entra en el estado de ejecución, donde imprime su cadena exactamente una vez antes de llamar a vTaskDelayUntil() para volver al estado Bloqueado.
1 - La tarea continua 1 se ejecuta durante un periodo de tictac completo (intervalo de tiempo entre los tiempos t1 y t2), durante el cual podría imprimir su cadena muchas veces.
5 - La tarea periódica que entra en el estado Bloqueado significa que el programador tiene que elegir de nuevo una tarea para que entre en el estado de ejecución; en este caso, se elige Continua 1 y se ejecuta hasta la siguiente interrupción de tictac, durante la cual podría imprimir su cadena muchas veces.
Periódico
Continua 1
Continua 2
Inactivo
La tarea inactiva nunca entra en el estado de ejecución, ya que siempre hay tareas de mayor prioridad que pueden hacerlo
t1
t2
t3
Tiempo
t5
2 - La interrupción de tictac ocurre durante la cual el programador selecciona una nueva tarea para ejecutar. Como ambas tareas Continuas tienen la misma prioridad y ambas siempre pueden ejecutarse, el programador comparte el tiempo de procesamiento entre las dos, por lo que Continua 2 entra en el estado En ejecución, donde permanece durante todo el periodo de tictac, tiempo durante el cual podría imprimir su cadena muchas veces.
3 - En el tiempo t3, la interrupción de tictac se ejecuta de nuevo, lo que provoca un cambio de vuelta a Continua 1, y así sucesivamente


1- La Tarea 1 se ejecuta primero, ya que tiene la prioridad más alta
3- La Tarea 1 se ejecuta de nuevo cuando la Tarea 2 reduce su propia prioridad por debajo de la prioridad de la Tarea 1, y así sucesivamente
Tarea 1
Tarea 2
Inactiva
La tarea inactiva nunca se ejecuta, ya que ambas tareas de la aplicación siempre pueden ejecutarse y siempre tienen una prioridad superior a la prioridad inactiva
t1
t2
Tiempo
2 - La Tarea 2 se ejecuta cada vez que la Tarea 1 establece la prioridad de la Tarea 2 como la más alta


La tarea 2 se ejecuta durante el resto del intervalo de tiempo
La tarea 1 vuelve a entrar en el estado Bloqueado, lo que permite que la tarea 2 continúe ejecutándose
Tarea 1 (alta, evento)
Tarea 2 (prioridad inactiva, continua)
Tarea inactiva (continua)
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
La tarea inactiva se ejecuta durante una iteración de su bucle de implementación y luego cede para permitir que el programador seleccione otra tarea
La tarea 1 abandona el estado Bloqueado y reemplaza a la tarea 2

Flujo de cambio de contexto en detalle

El flujo que mencionaste es correcto. Vamos a desglosarlo más detalladamente:

1. **Preparación del cambio:**
   - El planificador detecta que es necesario un cambio (Ej: por interrupción)
   - Accede al TCB actual mediante **pxCurrentTCB** para obtener información de la tarea
2. **Guardado del contexto:**
   - El registro a1 (SP) apunta a la pila de la tarea actual
   - Se guardan en la pila todos los registros de la CPU (a0-a15)
   - Se guarda PS (estado del procesador) y PC (contador de programa)
   - Se actualiza **pxTopOfStack** en el TCB para que apunte a donde quedó el SP
3. **Selección de la nueva tarea**:
   - Se recorren las listas de tareas listas usando **xStateListItem**
   - Se selecciona la tarea de mayor prioridad en estado Ready
   - Se actualiza **pxCurrentTCB** apuntando al TCB seleccionado
4. **Restauración del contexto**:
   - Se lee **pxTopOfStack** del nuevo TCB
   - Se carga este valor en el registro a1 (SP)
   - Se extraen de la pila los valores guardados previamente para:
     - Registros generales (a0-a15)
     - PS (estado del procesador)
     - PC (contador de programa)
   - Al restaurar el PC, el procesador salta automáticamente a la instrucción donde se quedó la tarea

Planificador de Procesos del ESP-IDF

Tareas - Estados - TCBs
Debe ser:
- **Preemptive** -> Interrumpir
- **Prioridad Fija** (en principio) -> La más alta
- **Con División de Tiempo (Time Slicing o Round Robin)** -> No monopolizar el procesador (tick)

Planificador con 2 núcleos

Planificación por Prioridad
- ¿Puede aca?
- ¿Está disponible?

Preemption Adaptado
Sin núcleo asignado -> Núcleo actual

Round Robin Multinúcleo (Best Effort)
Tarea saliente -> Final de su lista de prioridad

Recomendaciones y ¿Cómo decide el planificador?

Consideraciones Prácticas

Ejecución no secuencial: No se garantiza un orden de ejecución entre tareas de igual prioridad, especialmente si están en distintos núcleos.
Afinidad de núcleo: Las tareas pueden ser fijadas a un núcleo específico, afectando su comportamiento y posibilidad de ejecución.
Sincronización y prioridades: El sistema admite herencia de prioridad para evitar bloqueos por inversión de prioridades en recursos compartidos.

¿Cómo decide el planificador?
Como programador, cada uno define el comportamiento de las tareas influyendo directamente en las decisiones del planificador.

- Prioridad
- Afinidad por el núcleo
- Bloqueo
- Retardo temporal
- Suspensión