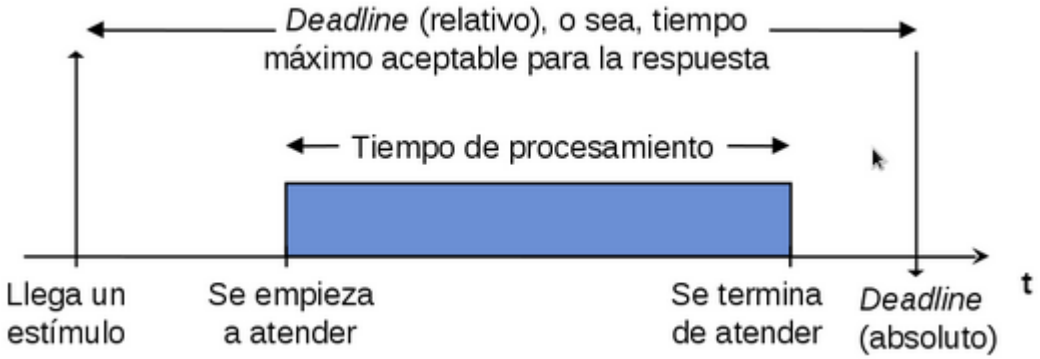
Sistemas Operativos de Tiempo Real (FreeRTOS)

Autores:
Isidro Buendía Ruz
Pablo Hinojosa Nava
Darío López Padial
Ricardo Villa-Real García-Valdecasas

Índice

1. Introducción

- Funcionamiento correcto en un tiempo determinado.
- No tiene porqué ser corto, si no acotado.
- Entradas, salidas y restricciones temporales conocidas.
- Sistema determinista, no hay nada aleatorio o que se escape del control del sistema.



1.1 Características Distintivas

- Uso en sistemas embebidos.
- Rápidos cambios de contexto, bajo intercambio entre almacenamiento secundario y memoria.
- Gestión de archivos orientada a la velocidad de acceso más que a la eficiencia.
- No son eficientes procesando información. Se usan procesadores predecibles.
- No existe la paginación, para evitar el factor aleatorio en la búsqueda.

1.2 Diferencias

- Tiempo de respuesta acotado.
- Objetivos muy limitados.
- Diseñados para soportar numerosas arquitecturas y montarse sobre muchos dispositivos.
- Máquinas de de características bajas (bajo procesamiento y poca memoria).

1.3 Aplicaciones

- Monitorización: máquinas de producción, aviones, automóviles.
- Centrales nucleares, robótica o sistema de procesamiento de transacciones.
- Telescopios de la NASA.

2. FreeRTOS

- Kernel de tiempo real totalmente gratuito
- Ofrece código abierto al ser software libre.
- Es el líder del mercado de RTOS, ya que es soportado por 31 arquitecturas de sistemas embebidos y con 77500 descargas al año, es el sistema operativo en tiempo real más ampliamente utilizado.

2.1 Primeros pasos freeRTOS

Ofrece las siguientes características:

- Diseñado para ocupar poca memoria, ser simple y fácil de usar.
- La estructura del código es muy portable.
- Permite el uso de procesos y corrutinas por separado o ambos a la vez.
- Ofrece un potente mecanismo de traza de ejecución y optimización.
- Facilitan aplicaciones de prueba (demos) preconfiguradas para el RTOS.
- Se puede considerar de tipo microkernel pues tan sólo nos ofrece un planificador y una API para comunicar procesos entre sí.

2.2 Gestión de planificación

Para que una función pueda ser manejada como un proceso debe tener el siguiente prototipo:

*void NombreDelProceso(void *pvParametros)*

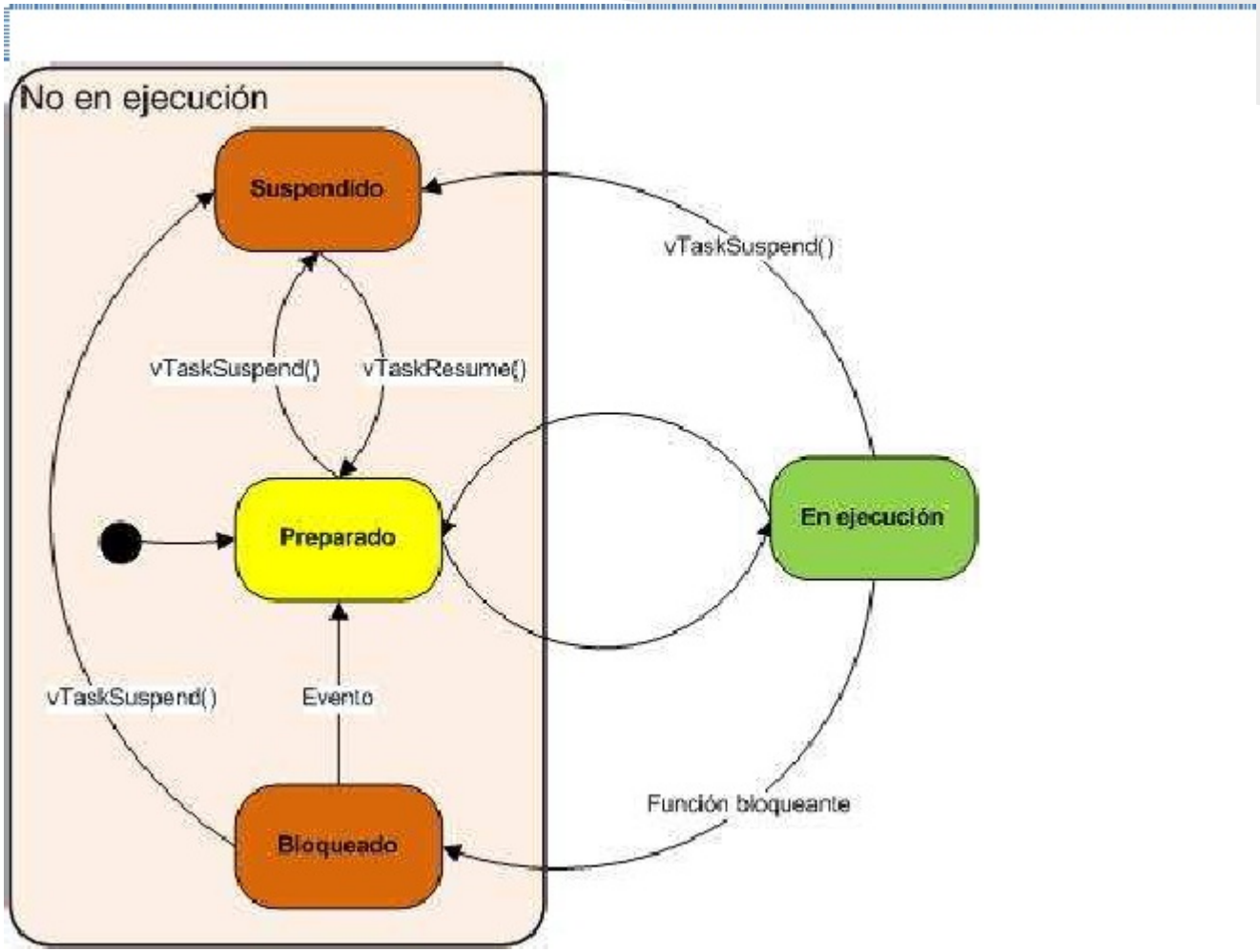
- Un proceso no debe salir de su función de implementación nunca, es decir, nunca contendrán una función return.
- Cada proceso es considerado como un programa independiente, siendo la primera función el punto de entrada que implemente el proceso.
- En un sistema con un solo procesador, como es el caso de la mayoría de sistemas embebidos, solamente una tarea podrá ejecutarse en un determinado instante de tiempo. Estas tareas podrán estar en dos estados principales: en ejecución o no en ejecución. A su vez, las que no están en ejecución, pueden estar en los siguientes estados: suspendida, bloqueada o preparada.

2.2 Gestión de planificación

Tareas suspendidas

Tareas bloqueadas

Tareas preparadas



2.3 Gestión de memoria

El kernel freeRTOS deberá asignar memoria principal (RAM) cada vez que se crea una tarea (su contexto de ejecución), una cola o un semáforo.

Para gestionar la memoria freeRTOS nos ofrece 3 esquemas de gestión de memoria que nos ofrecerán diferentes alternativas a la hora de decidir cuál escoger, pero también nos da la posibilidad de poder crearnos nuestro propio esquema para utilizarlo.

2.3 Gestión de memoria

Esquema 1:

Es el esquema más simple de todos. No nos permite liberar una zona de memoria una vez se haya asignado algo allí.
El algoritmo tan sólo asigna memoria del montículo, el cuál es dimensionado estáticamente en el archivo de configuración de FreeRTOS (FreeRTOSConfig.h) mediante la definición *configTOTAL_HEAP_SIZE*.

Este esquema:

- Es recomendable si la aplicación no elimina tareas o colas (no se realizan llamadas a vTaskDelete() o a vQueueDelete()).
- Es determinista ya que siempre tarda el mismo tiempo en devolver un bloque.

Este esquema es adecuado para una gran cantidad de RTOS que cumplan con la única condición de que todas las tareas y las colas se creen antes de que el kernel sea iniciado.

2.3 Gestión de memoria

Esquema 2:

Este esquema utiliza un mejor algoritmo de ajuste y nos permite liberar bloques de memoria que fueron asignados. Sin embargo, este esquema no ofrece ninguna reordenación de memoria, por lo que puede producirse fragmentación en la RAM.

Una vez más la cantidad total de memoria RAM disponible esta fijado por *configTOTAL_HEAP_SIZE* definido en el archivo FreeRTOS(FreeRTOSConfig.h).

Este esquema:

- Es recomendable para aplicaciones que reserven bloques de memoria RAM del mismo tamaño, es decir, tareas con el mismo tamaño de pila y colas de mensajes con el mismo tamaño.
- No es determinista.

Este esquema es adecuado para RTOS más pequeños que requieran crear dinámicamente las tareas.

2.3 Gestión de memoria

Esquema 3:

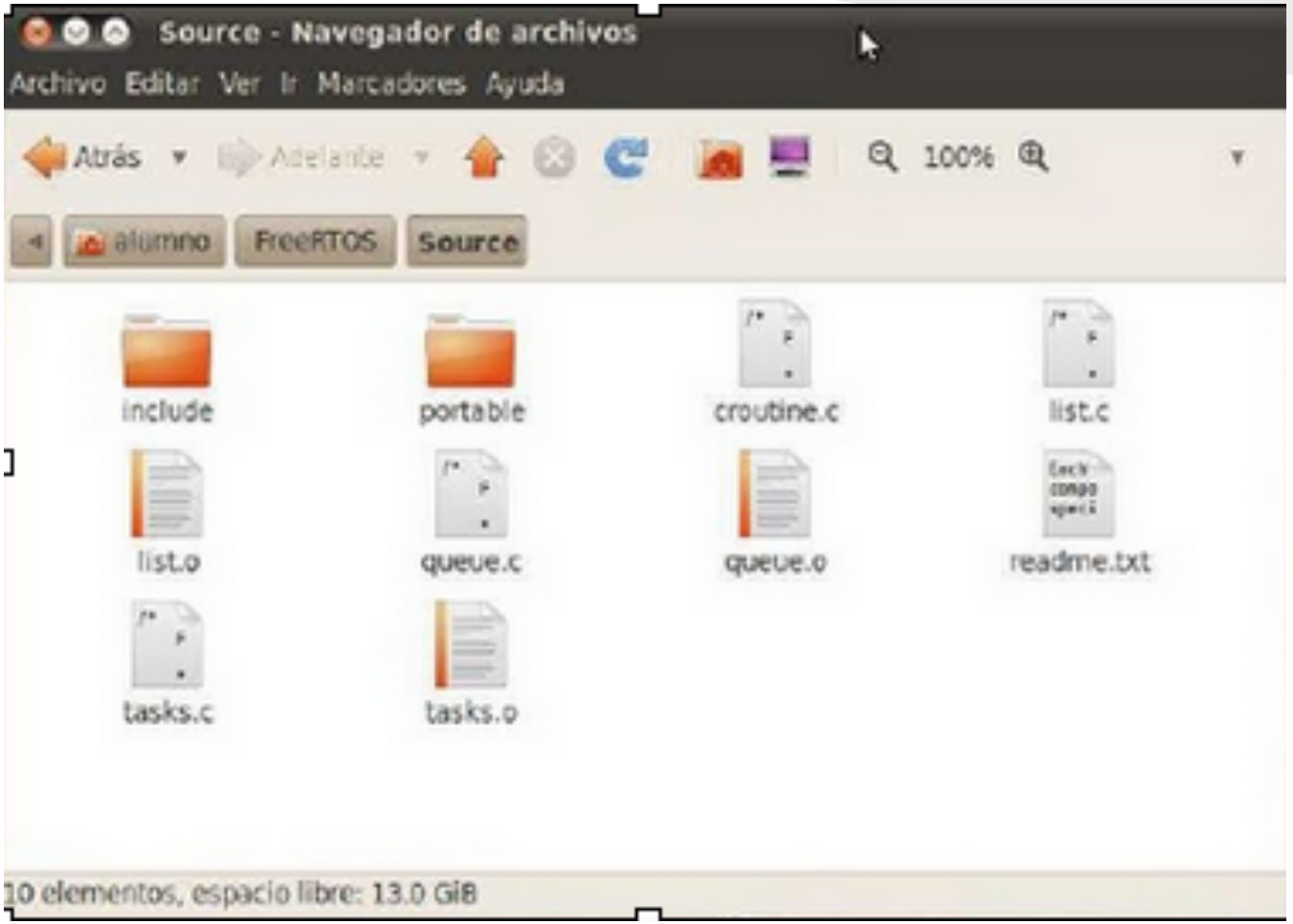
Este esquema es tan solo un contenedor para la utilización de las funciones estándar *malloc()* y *free()* con el objetivo de mantener seguros sus subprocesos ante posibles cambios de contexto.

Este esquema:

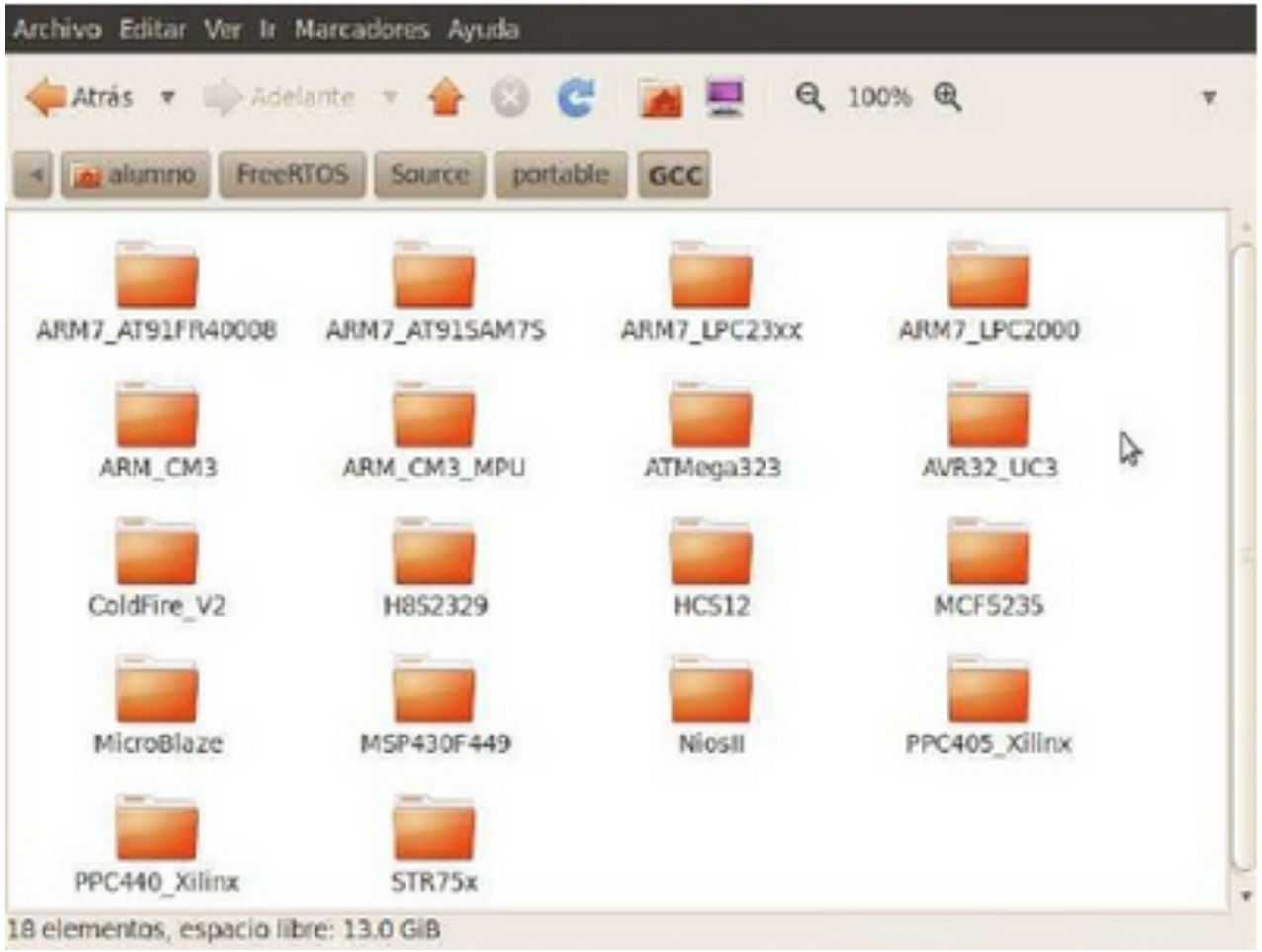
- Aumentará considerablemente el tamaño del código del kernel.
- No es determinista

Este esquema es el menos restrictivo y es el más adecuado para tomarlo como base
a la hora de crear nuestro propio esquema de gestión de memoria.

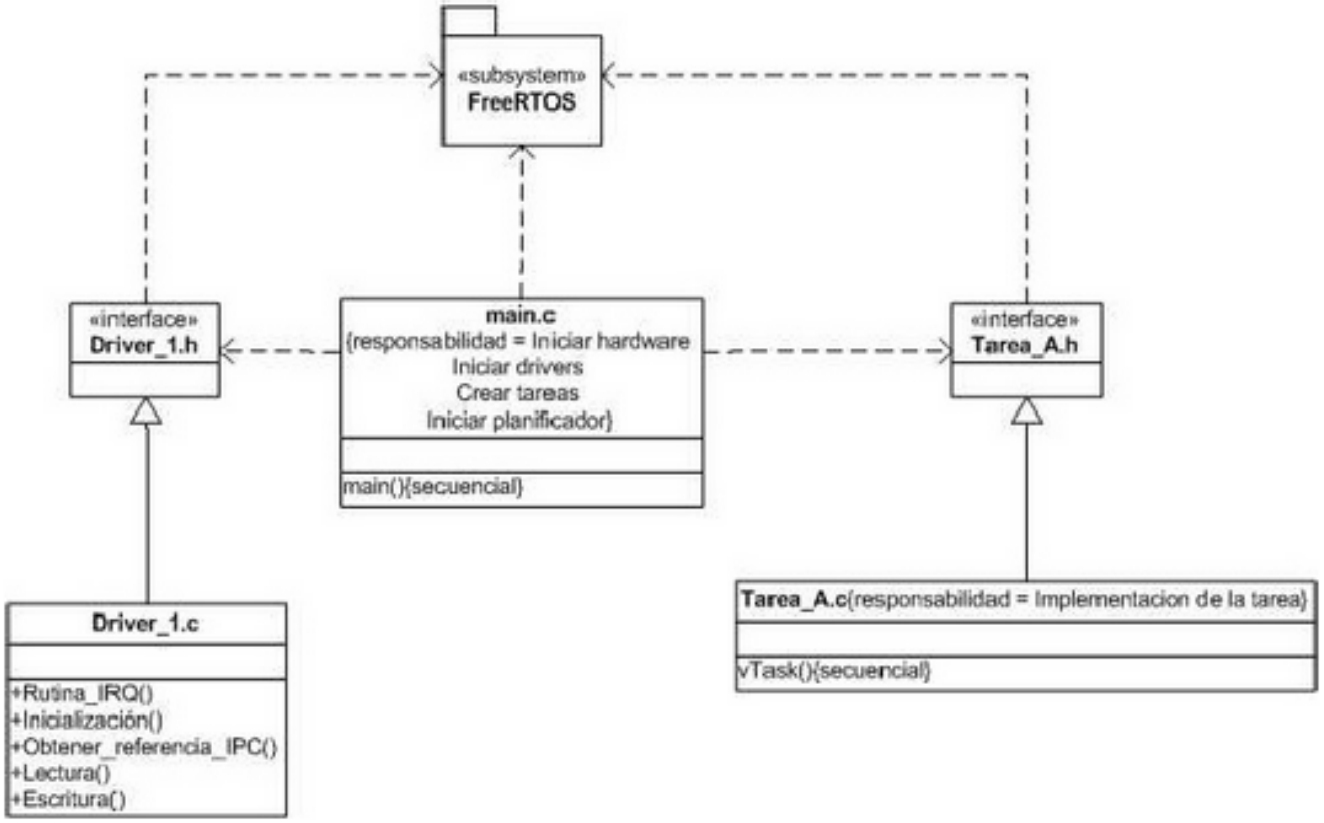
2.5 Estructura del código fuente de FreeRTOS


Source - Navegador de archivos
Archivo Editar Ver Ir Marcadores Ayuda
Atrás
Adelante
100%
alumno
FreeRTOS
Source
include
portable
croutine.c
list.c
list.o
queue.c
queue.o
readme.txt
tasks.c
tasks.o
10 elementos, espacio libre: 13.0 GiB


portable - Navegador de archivos
Archivo Editar Ver Ir Marcadores Ayuda
Atrás
Adelante
100%
alumno
FreeRTOS
Source
portable
BCC
CodeWarrior
GCC
IAR
Keil
MemMang
MPLAB
oWatcom
Paradigm
Renesas
Rowley
RVDS
SDCC
Softune
WizC
readme.txt
6 elementos, espacio libre: 13.0 GiB


Archivo Editar Ver Ir Marcadores Ayuda
Atrás
Adelante
100%
alumno
FreeRTOS
Source
portable
GCC
ARM7_AT91FR40008
ARM7_AT91SAM7S
ARM7_LPC23xx
ARM7_LPC2000
ARM_CM3
ARM_CM3_MPU
ATMega323
AVR32_UC3
ColdFire_V2
H8S2329
HCS12
MCF5235
MicroBlaze
MSP430F449
NiosII
PPC405_Xilinx
PPC440_Xilinx
STR75x
18 elementos, espacio libre: 13.0 GiB

2.5.1 Creación de un proyecto



2.5.2 Configuración y optimización

- FreeRTOS permite configurar un conjunto de parámetros para optimizar el sistema operativo
- Esta parametrización se hace en el archivo FreeRTOSConfig.h

2.5.2 Configuración y optimización

- **#define configUSE_PREEMPTION:** definiendo a 1 este parámetro configuramos el planificador para efectuar planificación expropiativa.
- **#define configUSE_CO_ROUTINES:** definiendo a 1 este parámetro habilitaremos la API de corrutinas.
- **#define configTICK_RATE_HZ:** en este parámetro indicamos la frecuencia del tick del sistema.
- **#define configTOTAL_HEAP_SIZE:** es el tamaño de memoria disponible para el kernel para el montículo.
- **#define configUSE_TRACE_FACILITY**: definiendo a 1 este parámetro podemos utilizar macros gancho para trazar la ejecución de FreeRTOS.
- **#define configCHECK_FOR_STACK_OVERFLOW**: este parámetro permite implementar un mecanismo para detectar desbordamientos de la pila.

3. Otros SO de tiempo real

Versiones de FreeRTOS

FreeRTOS: (GPL con excepción)

-Código del SO: Libre (GPL).

-Demos: Otras licencias.

OpenRTOS (versión comercial sin GPL)

-Incluye protección legal y ST profesional

SafeRTOS

-Cumple estándar IEC 61508

Empresa: High Integrity Systems

3. Otros SO de tiempo real

Ecosistema FreeRTOS+

FreeRTOS+Training: Curso de 3 días

FreeRTOS+IO: provee una interfaz de la librería de drivers periféricos tipo Linux/POSIX

FreeRTOS+CLI: provee una linea de comandos

FreeRTOS+Trace: provee una herramienta de diagnóstico

Nabto:provee conexión a internet

Embedded TCP/IP: servidor HTTP y Telnet

Ecosistema FreeRTOS+

CyaSSL: provee capa de seguridad

Safety Critical Certified: para aplicaciones críticas. Cumple certificados IEC 61508, EN62304 y FDA 510(k)

3. Otros SO de tiempo real

Alternativas a FreeRTOS

-RTLinux

-VXWorks

-QNX

Fin

Gracias por su atención :)