ANIOT. Práctica 2. Gestión de tareas en ESP-IDF

Patricia Barrios
Santiago Zomeño

Cuestiones, comentarios sobre el diseño, ejemplo de trazas, chuletas para compilación

---

1. Cuestiones

- **¿Qué prioridad tiene, por defecto, la tarea que se crea en el arranque para ejecutar *app_main()*?**

  La prioridad por defecto para *app_main()* es 3.

- **¿Qué prioridades has asignado a cada uno de las tareas que has creado?**

  · Las dos tareas sensor tienen prioridad: `PRIORITY_SENSOR_TASK 4`

  · Las dos tareas de filtrado tienen prioridad: `PRIORITY_FILTER_TASK 3`

  · La tarea controlador tiene prioridad: `PRIORITY_CONTROLLER_TASK 2`

- **¿Por qué es imprescindible asignar un *core* fijo a las tareas de tipo filtro?**

  En años anteriores sí que era necesario asignar un core fijo a cada tarea filtro, porque manipulan datos float.

  Resulta que cuando una tarea deja de estar en estado *running* (porque pasa por ejemplo a estado *suspended*), no guarda el estado de los registros de tipo float. Esto provoca que cuando la tarea reanude su ejecución, si lo hace sobre otro procesador, los valores float serán ya otros, generando un problema de inconsistencia. Asignando el mismo core se consigue continuar con los valores float anteriores.

  Ahora bien, este año, ya no es necesaria esta asignación explícita desde código, puesto que la versión actual del SO ha realizado esta corrección al respecto. Si observa algún dato de tipo de real en la tarea a ejecutar, le asignará siempre (automáticamente) el mismo core. Realizando por nosotros el "PinnedToCore"(*xTaskCreatePinnedToCore*).

  En nuestro ejercicio, a propósito, "no"hemos querido usar la instrucción *xTaskCreatePinnedToCore*. Queríamos dejar el código preparado para ser reutilizado en el futuro, y poder escalarlo a más sensores y filtros. Es por ello que está parametrizado a la medida, para que las tareas filtro que se fuesen generando, se ejecutasen en cualquiera de los cores. A partir de ahí el SO, o el hw (debidamente mejorado/corregido), son los que se encargarán (en su nivel de abstración) de asignarle el correspondiente procesador. En nuestro nivel de abstración queríamos usar sólo el *xTaskCreate*.

Aún aśi, a fecha de hoy, este problema está solventado (parcheado) sólo por sw (SO). Tal vez en el futuro se solvente via hw... Pero si todavía hubiésemos tenido que arrastrarlo y solventarlo, desde nuestro código, entendemos que al escalar sensores y filtros, hubiésemos tenido que aplicar algún mecanismo de reparto o discriminación de tareas entre los procesadores; por ej. tal vez un contador de tareas, asignando las tareas pares al core 0, y las tareas impares al core 2.

2. Comentarios sobre el diseño

- Menuconfig: Se consideró oportunó reflejar en el *menuconfig* las frecuencias de muestreo de nuestros sensores. En esta demo: SENSOR 0 (2000 ms) y SENSOR 1 (4000 ms); tal y como están definidos en *Kconfig.projbuild*. Además por requisito de la práctica y para poder usar la función *vTaskGetRunTimeStats*, era necesario activar en el *menuconfig* la opción *Component Config / FreeRTOS / Enable FreeRTOS to collect run time stats*.

- Headers: Tal y como ofrece la naturaleza de C, se ha implementado el correspodiente *types.h* que acompaña a nuestro *main.c*. En este *types.h* hemos recopilado principalmente las constantes de configuración (parametrización a la medida), y otras constantes y tipos usados en el código. Todo ello, con la finalidad de que a posteriori se reutilicen estos códigos.

- Parametrización: Se ha hecho incapié en aquellas variables que por intuición podrían resultar útiles a posteriori, cuando queramos reutilizar estos códigos, y ensamblar unas prácticas con otras en el proyecto final; por ej., el número de sensores, elementos de las colas, coeficientes, prioridades de las tareas, intervalos, ...

- printf comentados: Durante el desarrollo, resultaban muy útiles para las pruebas continuas, la secuenciación de *printf*, con los resultados parciales en ese momento y lugar del código. Aunque al final del ejercicio, sí o sí, hay que aligerar la traza que se genera, para no ser tan voluminosa y mostrar sólo lo necesario, no nos ha parecido mala idea mantener los *printf*, aunque comentados. Y es que de por sí ya estaban colocados en lugares estratégicos, y tal vez en futuros repasos y reutilizaciones, vengan de perlas poder descomentarlos rápidamente y ver de nuevo la traza local en ese punto.

- Idioma Inglés: aunque en un primer intento estaba todo el código expresado en español, nos pareció más adecuado, renombrar variables, constantes, comentarios, ... para que quedase en versión inglés. El objetivo de este hábito, es que estos códigos puedan ser entendidos, no sólo por hispanohablantes (que en su mayoría ya dominan el inglés técnico), si no que también por otras personas que provengan de otros idiomas. Quien sabe, muy fácilmente, el día de mañana pretendamos recuperar éste código, para añadirlo a un proyecto cuyo grupo de trabajo contempla personal de distintas nacionalidades (cosa cada vez más común).

3. Ejemplo de trazas

Véase aquí un ejemplo de trazas, resultado de la ejecución del ejercicio:

```
sensor_1.Generated data:93 (each 1000 ms) Thu Jan  1 00:00:01 1970
sensor_0.Generated data:10 (each 1000 ms) Thu Jan  1 00:00:01 1970
sensor_0.Generated data:5 (each 1000 ms) Thu Jan  1 00:00:02 1970
sensor_1.Generated data:55 (each 1000 ms) Thu Jan  1 00:00:02 1970
sensor_1.Generated data:18 (each 1000 ms) Thu Jan  1 00:00:03 1970
sensor_0.Generated data:33 (each 1000 ms) Thu Jan  1 00:00:03 1970
sensor_0.Generated data:55 (each 1000 ms) Thu Jan  1 00:00:04 1970
sensor_1.Generated data:79 (each 1000 ms) Thu Jan  1 00:00:04 1970
sensor_1.Generated data:48 (each 1000 ms) Thu Jan  1 00:00:05 1970
sensor_0.Generated data:3 (each 1000 ms) Thu Jan  1 00:00:05 1970
sensor_1.Controller queue.Getting:54.2
filter_sensor_1 - 54.2 - Thu Jan  1 00:00:05 1970
sensor_0.Generated data:35 (each 1000 ms) Thu Jan  1 00:00:06 1970
sensor_1.Generated data:85 (each 1000 ms) Thu Jan  1 00:00:06 1970
sensor_0.Controller queue.Getting:21.05
filter_sensor_0 - 21.05 - Thu Jan  1 00:00:05 1970
sensor_1.Generated data:27 (each 1000 ms) Thu Jan  1 00:00:07 1970
sensor_0.Generated data:33 (each 1000 ms) Thu Jan  1 00:00:07 1970
sensor_0.Controller queue.Getting:28.3
filter_sensor_0 - 28.3 - Thu Jan  1 00:00:06 1970
sensor_0.Generated data:40 (each 1000 ms) Thu Jan  1 00:00:08 1970
sensor_1.Generated data:52 (each 1000 ms) Thu Jan  1 00:00:08 1970
sensor_1.Controller queue.Getting:66.65
filter_sensor_1 - 66.65 - Thu Jan  1 00:00:06 1970
sensor_1.Generated data:66 (each 1000 ms) Thu Jan  1 00:00:09 1970
sensor_0.Generated data:90 (each 1000 ms) Thu Jan  1 00:00:09 1970
sensor_1.Controller queue.Getting:49.4
filter_sensor_1 - 49.4 - Thu Jan  1 00:00:07 1970
sensor_0.Generated data:97 (each 1000 ms) Thu Jan  1 00:00:10 1970
sensor_1.Generated data:70 (each 1000 ms) Thu Jan  1 00:00:10 1970
sensor_0.Controller queue.Getting:31.2
filter_sensor_0 - 31.2 - Thu Jan  1 00:00:07 1970
sensor_1.Generated data:64 (each 1000 ms) Thu Jan  1 00:00:11 1970
sensor_0.Generated data:89 (each 1000 ms) Thu Jan  1 00:00:11 1970
sensor_0.Controller queue.Getting:34.55
filter_sensor_0 - 34.55 - Thu Jan  1 00:00:08 1970
STADISTICS_INTERVAL/CONTROLLER_INTERVAL: 10000/1000 = 10

Task stadistics in cpu:
controller      47397           <1%
IDLE1           11000444                49%
IDLE0           10945834                49%
read_sensor_0   10073           <1%
filter_sensor_1 410             <1%
filter_sensor_0 400             <1%
read_sensor_1   10428           <1%
Tmr Svc         50              <1%
esp_timer       27              <1%
ipc1            9786            <1%
ipc0            4947            <1%
```

Listing 1: Ejemplo traza

4. Chuletas usadas: compilación

Como comentario final, tal vez no sea tan mala idea, divulgar y contar a los demás artimañas a las que recurrimos muchos de nosotros, como estudiantes durante los ejercicios. En este entorno, chuletas; truco idóneo para alumnos principiantes. En nuestro caso para las continuas y sucesivas compilaciones, volcados, trazas, etc... Y como en otros muchos contextos de programación, el premio se la lleva, la de las teclas rápidas...

```
COMPILACION NATIVA

cd /home/ubuntu/esp/esp-idf
. export.sh

cd /home/ubuntu/miproyecto
idf.py --version
idf.py menuconfig
idf.py build
idf.py -p /dev/ttyUSB0 flash
idf.py -p /dev/ttyUSB0 monitor
------------------------------------------------------------------------
COMPILAION VISUAL STUDIO CODE

pio run -t menuconfig
pio run
pio run -t upload
pio device monitor
------------------------------------------------------------------------
COMPILAION VISUAL STUDIO CODE (TECLAS RAPIDAS)

ctrl-alt-b
ctrl-alt-u
ctrl-j
ctrl-alt-s
F11
------------------------------------------------------------------------
COMPILAION VISUAL STUDIO CODE (MENU)

View --> Command Palette --> PlatformIO: Build
View --> Command Palette --> PlatformIO: Upload
View --> Command Palette --> PlatformIO: Serial Monitor
```

Listing 2: Chuleta para las distintas vías de compilación