Nano-Kernel Bare-Metal para RP2040

Informe de viabilidad

Matías Cajal

Universidad Nacional de Río Negro

Laboratorio de Sistemas Embebidos

5 de Abril de 2025

Índice

Introducción

El objetivo de este desarrollo es implementar un nano-kernel experimental ejecutándose directamente sobre el hardware (bare-metal) de un microcontrolador RP2040. Se busca explorar los conceptos fundamentales de los sistemas operativos de bajo nivel, como la gestión de tareas concurrentes, el manejo de interrupciones y el control directo de periféricos. La característica distintiva de este proyecto es su implementación manual desde cero, utilizando los lenguajes C y Assembler y basándose exclusivamente en la documentación técnica oficial del hardware, sin recurrir a Kits de Desarrollo de Software (SDKs) o bibliotecas estándar preexistentes.

Desarrollo

El sistema consistirá en un núcleo de software mínimo capaz de gestionar la ejecución concurrente de múltiples tareas mediante un scheduler pre-emptive simple y proporcionar una interfaz de usuario básica a través de una shell por puerto serie (UART).

Componentes y Funcionalidad Clave

- **Plataforma Hardware:** Se utilizará una placa Raspberry Pi Pico, basada en el microcontrolador RP2040 (Dual-Core ARM Cortex M0+).
- **Entorno:** El software se ejecutará en modo bare-metal, sin ningún sistema operativo subyacente.
- **Lenguajes:** La lógica principal, los drivers y la shell se desarrollarán en C (modo freestanding, sin libc). Rutinas críticas como el código de arranque y el cambio de contexto se implementarán en Assembler ARM Thumb.
- **Arranque:** Se desarrollará manualmente el código de arranque (`startup.S`) que se ejecuta al inicio, incluyendo la configuración de la tabla de vectores, la inicialización de la memoria RAM (copia de sección `.data`, limpieza de `.bss`) y la configuración inicial del puntero de pila, antes de saltar a la función `main` en C. Se creará también manualmente un Linker Script (`.ld`) para definir la ubicación de estas secciones en memoria.

- **Acceso a Hardware:** Toda interacción con los periféricos (GPIO, UART, Timer, etc.) se realizará mediante acceso directo a los registros de memoria mapeada. Las direcciones, offsets y máscaras de bits se definirán manualmente en archivos de cabecera (`.h`) a partir de la interpretación directa del datasheet del RP2040, utilizando punteros `volatile` en C.
- **Scheduler:** Se implementará en C un planificador simple que se activará periódicamente mediante la interrupción de un Timer del RP2040. Gestionará un conjunto de Tareas, cada una con su propio contexto y stack, utilizando una estructura de control de tareas (TCB) simple.
- **Cambio de Contexto:** El mecanismo para guardar el estado de la CPU de la tarea actual y restaurar el de la siguiente se implementará en Assembler, típicamente asociado a la interrupción del Timer o a la excepción PendSV para la pre-emption.
- **Drivers Bare-Metal:** Se escribirán en C las funciones necesarias para inicializar y operar los periféricos básicos requeridos: GPIO (control del LED onboard), UART (comunicación para la shell) y Timer (tick del scheduler), utilizando las definiciones manuales de registros. Se configurará el NVIC (Nested Vectored Interrupt Controller) para manejar las interrupciones necesarias.
- **Shell Interactiva:** Se implementará como una tarea gestionada por el scheduler. Utilizará el driver UART para leer comandos desde una terminal conectada vía un adaptador USB-Serie TTL, los parseará y ejecutará acciones básicas como leer/escribir memoria (`peek`/`poke`), controlar el LED (`gpio set`) o mostrar el estado de las tareas (`show-tasks`).
- **Tareas de Ejemplo:** Para demostrar la concurrencia, se ejecutarán al menos dos tareas: la Shell y una tarea simple adicional (ej. hacer parpadear el LED a un ritmo constante).

Motivación y Casos de Uso

La principal motivación detrás de este proyecto es de carácter educativo y exploratorio. El objetivo es desmitificar el funcionamiento interno de un microcontrolador y los bloques constructivos elementales de un sistema operativo embebido. Al prescindir deliberadamente de SDKs y bibliotecas de alto nivel, el desarrollo obliga a:

- Una comprensión profunda del datasheet del hardware.

- El manejo directo de la arquitectura del procesador (ARM Cortex-M0+), incluyendo su modelo de memoria y excepciones.
- La implementación manual de mecanismos de bajo nivel como el cambio de contexto y la gestión de interrupciones.
- La gestión explícita de recursos (memoria, tiempo de CPU).

Aunque el nano-kernel resultante no busca ser un producto robusto o de propósito general, sirve como una plataforma invaluable para experimentar y aprender. Los casos de uso se centran en el ámbito académico y de autoaprendizaje: como base para futuros experimentos con sistemas de tiempo real, como herramienta para entender la depuración a bajo nivel, o como ejercicio práctico en cursos de arquitectura de computadoras o sistemas embebidos. El conocimiento adquirido es directamente aplicable a cualquier entorno de desarrollo embebido, especialmente aquellos con recursos muy limitados o requerimientos de control muy precisos.

Viabilidad

Este proyecto presenta un desafío técnico considerable debido al enfoque sin SDK, que elimina las ayudas habituales en la configuración inicial y el acceso a periféricos. Sin embargo, se considera viable dentro del plazo del laboratorio (hasta el 16 de junio) bajo las siguientes condiciones:

- **Enfoque Incremental:** Se seguirá un plan de trabajo por fases, asegurando hitos funcionales mínimos (Blinky, UART TX, Shell monotarea) antes de abordar la complejidad del scheduler pre-emptive.
- **Plataforma Adecuada:** La Raspberry Pi Pico (RP2040) es una plataforma excelente para bare-metal debido a su buena documentación (datasheet) y arquitectura Cortex-M0+ relativamente sencilla, aunque se prescinda de su SDK. Además de la existencia de mucha información y proyectos de esta plataforma.
- **Recursos:** Económicamente es muy accesible. Se cuenta con la placa Pico y el adaptador USB-Serie TTL (CP2102 3.3V).

Los principales riesgos técnicos se asocian al código de arranque, el linker script, las definiciones manuales de registros y, fundamentalmente, en la rutina de cambio de contexto en Assembler, donde un error puede ser difícil de diagnosticar sin la capacidad de debuguear.

Componentes/Materiales Requeridos:

- Hardware: Raspberry Pi Pico, Adaptador USB-Serie CP2102 (3.3V), Cables USB.
- Software: Toolchain GNU ARM Embedded (`arm-none-eabi-gcc`, `as`, `ld`, `objcopy`), Editor de Texto/IDE, Software de Terminal Serie (ej. `minicom`, `tio`, PuTTY), Makefile.
- Documentación: Datasheet RP2040, ARM Cortex-M0+ Devices Generic User Guide.

Consideraciones

La dificultad principal reside en la ausencia total de abstracciones proporcionadas por un SDK. Cada interacción con un periférico requiere consultar el datasheet para determinar la secuencia exacta de lecturas/escrituras a registros específicos, definidos manualmente. La depuración dependerá fuertemente de la salida por el puerto serie (una vez que funcione) y de la observación del comportamiento. La implementación del cambio de contexto en Assembler requiere un entendimiento preciso del stack frame del Cortex-M0+ y del manejo de excepciones. Es crucial ser metódico y probar cada componente de forma aislada antes de integrarlo.

Conclusión

El desarrollo de este nano-kernel bare-metal representa una oportunidad excepcional para adquirir una comprensión práctica y detallada de los sistemas embebidos en su nivel más fundamental. A pesar de su alta dificultad técnica, el proyecto es viable como un ejercicio de aprendizaje intensivo, enfocándose en el proceso y los conocimientos adquiridos sobre la arquitectura y la programación de bajo nivel en C y Assembler.

Referencias

Aclaración: Si se está leyendo esto en pdf, es posible acceder a las mismas dando click.

- RP2040 Datasheet
- Cortex-M0+ Generic User Guide
- Proyecto rp-hal (Rust support for the “Raspberry Silicon” family of microcontrollers)
- Playlist Raspberry Pi Pico Bare Metal Programming