Reporte. Semáforo con Doble Núcleo

Unidad Académica Multidisciplinaria Mante (UAMM)

Arquitectura de Computadoras

Montoya Garza Luis Ángel

Hernández Ruiz Haydee Michelle

Rueda Martínez Alison Michelle

Silva Sánchez Yamilka Arely

7° “E”

M.C. López Piña Daniel

Ciudad Mante, Tamaulipas. Octubre del 2025

Introducción

El microcontrolador RP2040 se distingue por su arquitectura de doble núcleo. Esta práctica tiene como objetivo principal demostrar la capacidad de ejecución concurrente de tareas utilizando el módulo _thread de MicroPython, que permite asignar código al segundo núcleo (Core 1). A través de la división del trabajo, se busca asegurar que la tarea principal de temporización (secuencia de LEDs en el Núcleo 0) y una tarea reactiva (monitoreo del pulsador en el Núcleo 1) se ejecuten de manera totalmente independiente y sin interrupciones mutuas. Esto es crucial para desarrollar sistemas embebidos donde se requiere alta capacidad de respuesta y precisión temporal en múltiples procesos simultáneos.

Desarrollo

Esta práctica implementa la multitarea concurrente en tu placa RP2040 (como la Raspberry Pi Pico) al dividir la carga de trabajo en sus dos núcleos de procesamiento. El objetivo principal es ejecutar dos tareas totalmente independientes y con temporizaciones distintas sin que una bloquee a la otra. Para lograr esto, el código asigna una tarea de secuencia de LEDs al Núcleo 0 (Principal) y la tarea de monitoreo del pulsador al Núcleo 1 (Secundario).

El código comienza con la importación de los módulos esenciales: `Pin` para el manejo de entradas y salidas, `sleep_ms` para las demoras, y fundamentalmente, `_thread`, que es la herramienta de MicroPython para gestionar hilos, permitiendo la ejecución de código en el segundo núcleo del chip RP2040.

Se definen todos los pines GPIO. El **pulsador** (`boton` en el pin 20) se configura como una entrada con resistencia *pull-down* interna, asegurando un estado bajo (0) por defecto. El **LED azul** (pin 19) y la secuencia de LEDs (Rojo en 14, Amarillo en 15, Verde en 16) se configuran como salidas. Se agrupan los LEDs de la secuencia en la tupla `leds` para facilitar la iteración posterior en el bucle principal.

La función estado_pulsador_thread() contiene el código que se ejecutará en el Núcleo 1. Esta función se mantiene en un bucle infinito, dedicada exclusivamente a verificar el estado del pulsador. Si el botón está presionado (boton.value() == 1), el LED azul se conmuta (azul.toggle()) cada 200 ms, creando un parpadeo. En

contraste, si el botón no está presionado, el LED azul simplemente permanece encendido (azul.value(1)), usando un retardo muy corto de 10 ms para evitar que el bucle consuma todos los ciclos de la CPU innecesariamente.

El nuevo hilo es iniciado inmediatamente después con la línea clave: _thread.start_new_thread(estado_pulsador_thread, ()). Esto envía la función estado_pulsador_thread al segundo núcleo y comienza su ejecución paralela de forma inmediata.

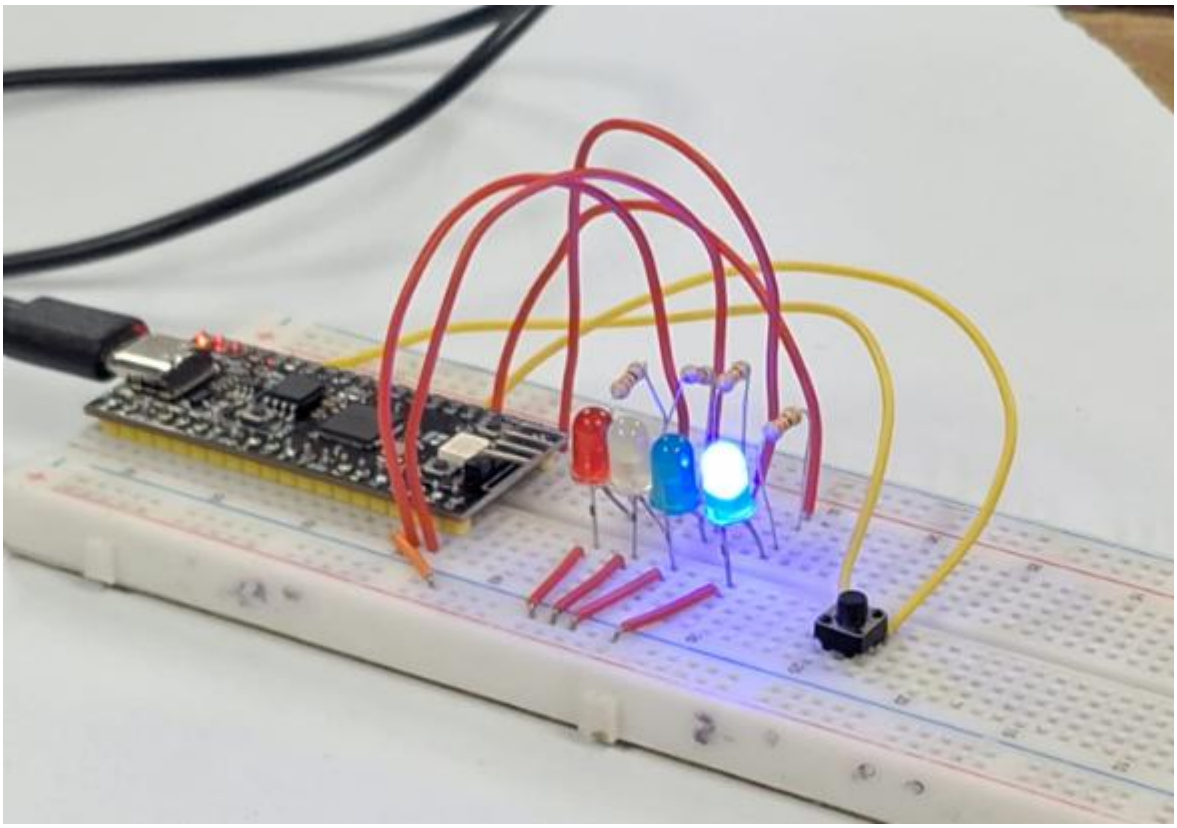
El resto del código, que es el bucle principal while 1, se ejecuta por defecto en el Núcleo 0. Esta tarea implementa una secuencia de LEDs tipo semáforo. Itera a través de los LEDs definidos en la tupla leds. Para cada LED, lo enciende por 500 ms y luego lo apaga por otros 500 ms antes de pasar al siguiente. Esta secuencia tiene una temporización de 1 segundo por LED.

Codigo:

```
from machine import Pin
from time import sleep_ms
import _thread

boton=Pin(20, Pin.IN, Pin.PULL_DOWN)
azul=Pin(19, Pin.OUT)

verde= Pin(16, Pin.OUT)
amarillo= Pin(15, Pin.OUT)
rojo= Pin(14, Pin.OUT)

leds=(rojo, amarillo, verde)

def estado_pulsador_thread():
    while 1:
        if boton.value()==1:
            azul.toggle()
            sleep_ms(200)
        else:
            azul.value(1)
            sleep_ms(10)

_thread.start_new_thread(estado_pulsador_thread, ())

while 1:
    for led in leds:
        led.value(1)
        sleep_ms(500)
        led.value(0)
        sleep_ms(500)
```

Evidencia

https://github.com/luisangelmg11/Arquitectura-de-Computadoras/blob/main/1.5%20Practica%20Evidencia%20Video.mp4



Conclusión

La implementación demostró exitosamente la ejecución en paralelo de dos procesos. El Núcleo 0 mantuvo una secuencia de LEDs con una temporización fija y predecible, mientras que el Núcleo 1 manejó eficientemente el monitoreo del pulsador y el parpadeo del LED azul. El uso del _thread.start_new_thread() validó la capacidad de la RP2040 para desglosar cargas de trabajo, lo que es esencial para evitar el bloqueo del programa principal por tareas de entrada/salida o de tiempo crítico. En resumen, la práctica confirma que la programación dual es una estrategia robusta y necesaria para optimizar el rendimiento y la capacidad de respuesta en aplicaciones complejas del RP2040.

Perfiles GitHub

https://github.com/luisangelmg11
https://github.com/HaydeesitaSJJH
https://github.com/alisonrueda
https://github.com/Yamixjk