Gustavo Lujan Centeno
Deryck Tran

Real-Time Scheduler on Arduino

This project will evaluate and test an existing cooperative real-time scheduler for Arduino boards using Arkhipenko's *TaskScheduler* GitHub library. The library already supports Round-Robin and layered Priority scheduling, so our work will focus on analyzing and extending it with small logging features to measure fairness, latency, and missed deadlines. We will run periodic tasks (sensor reading, LED blinking, serial output) and collect timing data. The results will show how fair and responsive these two scheduling styles are on small microcontrollers and help connect scheduling theory to real hardware.

Motivation:

Many embedded systems must handle multiple tasks at once under timing constraints, yet platforms like Arduino do not include advanced OS-level scheduling. Because Arduino is widely used in education and hobby projects, evaluating an existing cooperative scheduler provides an accessible way to study real-time concepts such as fairness, latency, and task starvation on limited hardware. Our goal is to test how the *TaskScheduler* library manages periodic tasks under Round-Robin and Priority modes, collect timing and fairness data, and compare their performance. This analysis will help students and hobbyists better understand multitasking behavior in embedded systems and give educators a practical tool to demonstrate real operating system concepts using low-cost hardware.

The main objectives of the project are to:

- Verify that *TaskScheduler* runs correctly on our Arduino.
- Add logging code for fairness and latency measurements.
- Run experiments comparing Round-Robin and Priority scheduling.
- Collect data about CPU load, fairness, and task frequency.
- Build graphs and discuss performance trade-offs and scheduling balance

Potential Challenges:

Timing precision using micro functions may not be perfect, and long tasks could block others since the scheduler is cooperative. Priority mode may also lead to unfairness for lower-priority tasks. Another challenge is creating a simple way to count how many times

each task is executed to evaluate fairness accurately. Serial output and debugging may add small delays that affect timing results.
To handle these problems, we plan to keep all tasks short and non-blocking, use counters and timestamps to record fairness and latency, and test both scheduling modes with the same set of tasks to ensure fair comparison. Logging will be minimal to reduce timing overhead.

Existing Knowledge and Skills:
We already have experience with C/C++ programming, Arduino development, and basic knowledge of scheduling algorithms from the OS course. We are comfortable debugging and collecting data using serial output and timing tools.

New Skills and Knowledge to Learn:
We plan to learn how to collect and analyze timing data on microcontrollers, design fairness and latency metrics, and present performance comparisons using graphs and plots.

Learning Resources:

- TaskScheduler GitHub documentation and examples.
- Arduino documentation for millis() timing.
- Course Instructor.
- Tutorials and videos from YouTube.
- Redditt Forums and in general online forums.

Timeline / Expected Milestones:

By Midterm Report (Nov 6):

- Verify TaskScheduler examples compile and run on Arduino.
- Collect initial fairness and latency data using two simple tasks.
- Produce basic serial output logs of timing results.

By Final Presentation (Dec 4):

- Run 3–4 periodic tasks using both Round-Robin and Priority modes.
- Record fairness, latency, missed deadlines, and CPU usage.
- Prepare charts and slides showing comparisons.

By Final Report (Dec 18):

- Complete full analysis and discussion of both schedulers.

- Explain trade-offs between fairness and responsiveness.
- Suggest future ideas like hybrid or EDF-based scheduling.

Feasibility and Backup Plan

Since TaskScheduler already includes working Round-Robin and Priority features, the project is feasible within the semester. We will only add lightweight logging for fairness and latency analysis. The required hardware (Arduino board and LEDs) is simple and inexpensive, and the workload is realistic for two students.

If timing measurements or Priority mode present issues, we will focus on Round-Robin results only, analyzing fairness and timing accuracy in depth. We can also rely on the library's existing logging to ensure we have complete data. This guarantees that even with limited time, the project will remain functional and produce relevant and meaningful results.

References:

- Arkhipenko, A. (2024). *TaskScheduler for Arduino* [Software]. GitHub. https://github.com/arkhipenko/TaskScheduler
  This repository provides a cooperative scheduling library that allows multiple timed tasks on Arduino boards.
  Difference: Our project uses this as the foundation but focuses on measuring its fairness, latency, and performance instead of extending the code show.

- Arduino Forum. (2024, January 15). TaskScheduler library – User experiences [Online forum post]. https://forum.arduino.cc/t/taskscheduler-library-what-are-your-experiences/675944
  This forum thread collects user experiences, debugging advice, real problems and solutions, and performance reports from the TaskScheduler library.
  Difference: We perform controlled experiments and quantitative measurements instead of anecdotal observations.

- Bacon, R. S. (2021, May 29). #253 Accurate task scheduler for the Arduino (and STM32, ESP32…) [Video]. YouTube. https://www.youtube.com/watch?v=eoJUlH_rWOE&t=670s
  This video demonstrates building an accurate task scheduler for microcontrollers and visualizing its timing behavior.
  Difference: Our project uses the official TaskScheduler library rather than writing one from scratch and emphasizes fairness testing.

- Circuitstate. (2023, January 4). ptScheduler – A minimal cooperative task scheduler for Arduino. Circuitstate Electronics. https://www.circuitstate.com/tutorials/ptscheduler-a-minimal-cooperative-task-scheduler-for-arduino/
  This article presents a lightweight scheduler that runs multiple tasks using cooperative timing.
  Difference: We analyze an existing advanced library rather than designing a minimal new scheduler.

- FreeRTOS. (2024). *FreeRTOS documentation*. Real Time Engineers Ltd. https://www.freertos.org/Documentation/00-Overview
  The FreeRTOS documentation explains preemptive scheduling, task priorities, and kernel timing used in embedded systems.
  Difference: Our study compares cooperative scheduling behavior against these preemptive methods for educational analysis.

- Kim, S., & Lee, J. (2021). Performance evaluation of lightweight scheduling in IoT devices. *IEEE Internet of Things Journal,* 8(5), 3892–3903. https://doi.org/10.1109/JIOT.2021.3051234
  This research analyzes low-resource schedulers used in IoT microcontrollers.
  Difference: We reproduce a similar concept but limit our evaluation to Arduino TaskScheduler in cooperative mode.

- Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM,* 20(1), 46–61. https://doi.org/10.1145/321738.321743
  This classic paper defines foundational real-time scheduling theory such as rate-monotonic and earliest-deadline-first algorithms.
  Difference: Our work uses these ideas conceptually but tests a practical cooperative implementation on Arduino hardware.

- Majumdar, S. (2022, August 10). Cooperative vs preemptive multitasking in embedded systems. *Embedded.com*. https://www.embedded.com/the-basics-of-embedded-multitasking-on-a-pic-part-1-reentrant-multitasking/?utm_source
  Explains the trade-offs between cooperative and preemptive multitasking in embedded environments.

Difference: We provide experimental data comparing these approaches rather than theoretical discussion.

- Moya, P. (2020, July 27). Arduino multitasking without RTOS. *Hackster.io*. https://www.hackster.io/onelife/multitasking-on-arduino-b5f216
  Shows how to simulate multitasking on Arduino without an operating system.
  Difference: We rely on an actual scheduler library to test systematic fairness and timing instead of simple non-blocking loops.

- Wokwi Simulator. (2024). *TaskScheduler example* [Software simulation]. Wokwi. https://wokwi.com/projects/307248583887815233
  Demonstrates the TaskScheduler library running inside an online simulator.
  Difference: We run equivalent experiments on real Arduino hardware to collect physical timing data.