Project descritption
Stefano Di Carlo authored 3 hours ago

99fa440e

Project-Assignment-2026.md 11.59 KiB

Projects

The project assignment for 2026 focuses on the development of new scheduling algorithms for FreeRTOS

🕒 Project 1 — Precise Scheduler for FreeRTOS

> A deterministic, timeline-driven scheduler replacing the default FreeRTOS priority-based model.

📘 Table of Contents

🧭 Overview

This project aims to **develop a precise, timeline-based scheduler** that **replaces the standard priority-based FreeRTOS scheduler**.
The scheduler enforces **deterministic task execution** based on a **major frame and sub-frame structure**, following the principles of **time-triggered architectures**.

Each task runs at **predefined times** in a global schedule, ensuring predictable real-time behavior and repeatability — without relying on dynamic priority changes.

⚙️ Key Features

1. Major Frame Structure

- The scheduler operates within a **major frame**, with a duration **defined at compile time** (e.g., 100 ms, 1 s).
- Each major frame is divided into **sub-frames**, smaller time windows that host groups of tasks.
- Each sub-frame can **contain multiple tasks** with assigned timing and order.

Example

```
Major frame = 100 ms
Sub-frames = 10 × 10 ms
→ The scheduler repeats this 100-ms timeline cyclically.
```

2. Task Model : NO PERIODIC TASKS

with no preemption

- Each task is implemented as a **single function** that executes from start to end and then terminates.
- Tasks do **not** include periodicity or self-rescheduling logic — the scheduler controls activation timing.
- If two tasks need to communicate, they do it through polling → NO SEMAPHORES, NO PREEMPTION

3. Task Categories

🧱 Hard Real-Time (HRT) Tasks

- Se ho già task RUNNING ⇒ kill per eseguire nuova task HRT
- Se non termina entro lo slot ⇒ Kill task HRT

- Assigned to a specific sub-frame.
- The scheduler knows the **start time** and **end times** of each task within the sub-frame.
- A task is **spawned at the beginning** of its slot and either:
  - runs until completion, or
  - is **terminated** if it exceeds its deadline.
- **Non-preemptive:** once started, it cannot be interrupted.

Example

```
Task_A → Sub-frame 2 (20-30 ms)
Start = 21 ms, End = 27 ms
→ Finishes at 26 ms → OK
→ Still running at 27 ms → Terminated
```

🌿 Soft Real-Time (SRT) Tasks → policy FIFO

- Executed **during idle time** left by HRT tasks.
- Scheduled in a **fixed compile-time order** (e.g., Task_X → Task_Y → Task_Z).
- **Preemptible** by any hard real-time task.
- **No guarantee** of completion within the frame.

Example

```
Sub-frame 5 (40-50 ms)
Task_B (HRT) uses 4 ms → Remaining 6 ms used by SRT tasks (Task_X → Task_Y → Task_Z)
```

4. Periodic Repetition

- At the end of each major frame:
  - All tasks are **reset and reinitialized**. This can be done by destroying and recreating all tasks or by resetting their state to an initial value.
  - The scheduler **replays the same timeline**, guaranteeing deterministic repetition across frames.

5. Configuration Interface

All scheduling parameters (start/end time, sub-frame, order, category, etc.) are defined in a **dedicated OS data structure**.

Example → ① Definisco struct per caratteristiche task ② Chiamo "startScheduler" → legge la struct, crea la task e la esegue (↳ NON eseguo task con metodo "executeTask" diretto. devo solo eseguire lo scheduler)

```c
typedef struct {
    const char* task_name;
    TaskFunction_t function;
    TaskType_t type; // HARD_RT or SOFT_RT
    uint32_t ulStart_time;
    uint32_t ulEnd_time;
    uint32_t ulSubframe_id;
} TimelineTaskConfig_t;
```

Selecting the most suitable structure to represent all tasks is a key part of the project assignment.

A system call (e.g. `vConfigureScheduler(TimelineConfig_t *cfg)`) will:

- Parse this configuration,
- Create required FreeRTOS data structures,
- Initialize the timeline-based scheduling environment.

🕒 Project 2 — Priority-Based Scheduler for Periodic Tasks in FreeRTOS

> **Goal:** Add first-class periodic tasks (period + deadline) **on top of** the default FreeRTOS scheduler, preserving FreeRTOS's preemptive, priority-based semantics.

🧭 1) Overview

FreeRTOS schedules tasks by priority but does not enforce **periodicity** or **deadlines**. This project introduces a thin **Periodic Task Layer (PTL)** that:

- lets users **declare periodic tasks** with *period* and *deadline*,
- performs **job releases** at the correct times,
- detects **deadline misses** and **period overruns**,
- and leaves **preemption** to FreeRTOS (unchanged).

> The PTL must patch the kernel scheduler with minimal intrusivity to make it easily portable.

📚 2) Terminology

- **Period (T):** time between releases.
- **Deadline (D):** relative deadline from each release. If not specified, **D = T**.
- **Release time (R_k):** k-th activation time; if all tasks start together, $R_0 = t_0$ for all.
- **Finish time (F_k):** time when job k completes.
- **Deadline miss:** `F□ > R□ + D`.
- **Overrun (period overrun):** previous job not finished at `R□□□` (i.e., exceeds T).

⚙️ 3) Functional Requirements

1. **Introduce dedicated periodic tasks API.** To create and handle tasks with `(period, deadline, priority, stack, name, entry)`.
2. **All tasks start together** (project requirement). *Optional*: allow a phase/offset parameter; if omitted, all start at t_0.

🕒 Project 3 — Priority-Based Scheduler for Periodic Tasks with Polling Server in FreeRTOS

> **Goal:** Extend the periodic task scheduler developed in *Project 2* by adding a **polling server mechanism** to manage **aperiodic tasks** within a predictable real-time framework.

🧭 1) Overview

In this project, you will implement a **Polling Server** on top of the **Priority-Based Periodic Task Scheduler** developed in Project 2.
The polling server acts as a **special periodic task** that is periodically activated to execute **aperiodic (event-driven) tasks** without violating the temporal constraints of hard real-time periodic tasks.

The system must include a **dedicated API** to register and manage aperiodic tasks.
Each aperiodic task will be associated with a **soft deadline**, defined relative to its **activation time** (i.e., the time when it is released for execution by the server).

If an aperiodic task exceeds its assigned deadline, the scheduler must apply one of two configurable policies:

- **OVERRUN:** allow the task to continue execution beyond its soft deadline; the deadline miss is logged, but the task completes normally.
- **KILL:** immediately terminate the aperiodic task upon a deadline overrun and proceed to the next one in the queue.

In both cases, the system must **log all deadline misses** with precise timestamps and task identifiers for trace and analysis purposes.

Assignment for all projects

Non-Functional Requirements

- **Release jitter:** ≤ 1 tick.
- **Overhead:** ≤ 10% CPU (≤8 tasks).
- **Portability:** QEMU Cortex-M. → Non è necessario essere specifici
- **Thread safety:** use FreeRTOS-safe synchronization.
- **Documentation:** clear API and timing model.
- **Codign guidelines** strictly follow freeRTOS coding guidelines.

Error Handling & Edge Cases

Handle all errors with dedicated hook functions following the FreeRTOS style.

Trace and Monitoring System

A **trace module** must be developed for test and validation purposes to log and visualize scheduler behavior with **tick-level precision**.

Logged information:

- Task start and end ticks
- Deadline misses or forced terminations
- CPU idle time

Example Output

```
[ 21 ms ] Task_A start
[ 26 ms ] Task_A end
[ 40 ms ] Task_B start
[ 47 ms ] Task_B deadline miss → terminated
```

```
[   0] A RELEASE
[   0] A START
[   5] A COMPLETE
[  10] A RELEASE
[  10] B RELEASE
[  10] A START
[  12] B START
[  25] B DEADLINE_MISS (D=15 @ tick 25)
[  30] A OVERRUN → SKIP
[  40] A RELEASE
...
```

Production and Regression Testing

Develop an **automated test suite** to validate the scheduler's correctness and robustness.

The suite must include:

- Stress tests (e.g., overlapping HRT tasks).
- Edge-case tests (e.g., minimal time gaps).
- Preemption and timing consistency checks.

Each test must:

- Produce **human-readable summaries**, and
- Include **automatic pass/fail checks** for regression testing.

Example

```
Test 3 - Overlapping HRT Tasks: FAILED (Task_A killed at 32 ms)
Test 4 - SRT Preemption: PASSED
```

Configuration framework (optional)

Create a high-level configuration framework (e.g., a Python script) that enables the user to describe the target problem, perform schedulability analysis (for algorithms that support it), and ultimately generate the FreeRTOS skeleton application.

📦 Deliverables

1. ✅ Modified FreeRTOS kernel with timeline-based scheduler.
2. ✅ Configuration data structure and system call for schedule definition.
3. ✅ Trace and monitoring system with tick-level resolution.
4. ✅ Automated test suite for validation and regression checking.
5. ✅ Documentation and example configurations.

🕒 Development Tools and Licenses

It is mandatory to use GitLab for all development purposes. A GitLab account for each group member will be created soon. You will receive an email with credentials.

Every file must be released following the FreeRTOS license schema.