ISTANBUL TECHNICAL UNIVERSITY

Computer Engineering Department

Real-Time Systems Software (RTSS)
Project Assignment

Real-Time Scheduling Simulator

Prepared by:
Name Surname: Batuhan KARAKUŞ
Student ID: 150210040

Course: BLG 450E - Real-Time Systems Software
Instructor: Prof. Dr. Deniz Turgay Altılar

December 1, 2025

Contents

Abstract

This report presents the design and implementation of a comprehensive **Real-Time Scheduling Simulator**. The software simulates fundamental and advanced scheduling algorithms, including *Rate Monotonic (RM)*, *Deadline Monotonic (DM)*, *Least Laxity First (LLF)* and *Earliest Deadline First (EDF)*. It features a multi-core scheduling engine supporting global scheduling logic. To handle aperiodic tasks, advanced server mechanisms such as *Poller*, *Deferrable Server*, and *Sporadic Server* are implemented. The application includes a custom "Task Creator" interface for both manual input and smart random task generation. Additionally,The simulator is also deployed as a web application for cross-platform accessibility. You can access and run the simulator directly via the following link: **Click here to launch RTSS Web Simulator**

1 Introduction

Real-time systems require strict adherence to timing constraints. The objective of this project is to develop a simulator that allows users to analyze the schedulability of task sets under various algorithms and processor configurations. The tool provides a visual representation of the schedule, identifies deadline misses, and calculates system utilization (U).

2 System Model and Assumptions

The simulator operates under standard real-time theory assumptions:

- **Periodic Tasks (P_i):** Defined by release time (r_i), execution time (C_i), period (T_i), and deadline (D_i).
- **Aperiodic Tasks (A_i):** Tasks with arbitrary arrival times.
- **Server Tasks (S_i):** Special periodic tasks aimed at servicing aperiodic requests (Bandwidth Preserving Servers).

2.1 Schedulability Metrics

System utilization (U) is dynamically calculated based on the number of cores (M):

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}, \quad \text{Capacity} = M \times 1.0 \tag{1}$$

The system warns the user if $U >$ Capacity.

3 Implemented Algorithms

The simulator supports the following algorithms:

3.1 Priority Assignment

- **Rate Monotonic (RM):** Static priority based on periods (T_i).
- **Deadline Monotonic (DM):** Static priority based on relative deadlines (D_i).
- **Earliest Deadline First (EDF):** Dynamic priority based on absolute deadlines ($d_i(t)$).
- **Least Laxity First (LLF):** Dynamic priority based on task laxity ($L_i(t)$), where tasks with the smallest laxity are executed first.

3.2 Aperiodic Server Mechanisms

- **Background:** Aperiodic tasks run only when the processor is idle.
- **Poller:** Budget is available at period start but lost immediately if no work exists.
- **Deferrable Server:** Budget is preserved throughout the period.
- **Sporadic Server:** Budget replenishments are dynamic, occurring T_s time units after consumption.

4 Software Design and Implementation

The project is implemented using an Object-Oriented approach in **Python**.

4.1 Simulation Engine

The `run_simulation` function serves as the core engine. It supports **Global Scheduling** for multi-core systems.

1. **Time Loop:** Iterates from $t = 0$ to LCM.
2. **Arrival Check:** Handles Periodic, Server, and Aperiodic arrivals.
3. **Replenishment Logic:** Specifically complex for Sporadic Server (replenishment queue).
4. **Priority Queue:** Sorts ready jobs based on the selected algorithm.
5. **Dispatching:** Assigns top M jobs to M cores.

4.2 Task Generator Module

A "Smart Random Generator" is implemented to create feasible task sets. It uses a logic similar to the *UUniFast* algorithm to distribute utilization randomly among tasks and allows the user to inject a Server task automatically.

4.3 Deployment Strategy

To ensure accessibility across different operating systems, the application is deployed in two formats:

- **Desktop Application:** A standalone executable built with Tkinter.
- **Web Application:** A cloud-based version deployed using Streamlit (accessible via browser).

5 User Interface and Experimental Results

The GUI is designed with a modern "Dark Theme" and "Flat Design" principles.

5.1 Features

- **Task Creator Window:** Separate tabbed interface for Manual Entry and Random Generation.
- **Interactive Gantt Chart:** Users can hover over task blocks to see detailed information (Tooltip).
- **Smart Visualization:** The chart automatically switches between "Task-Centric View" (Single-core) and "Core-Centric View" (Multi-core).
- **Reporting:** One-click export generates a detailed `.txt` report and a high-resolution `.png` chart.

Figure 1: Main Interface showing Multi-core EDF Scheduling

6 Conclusion

The developed Real-Time Scheduling Simulator successfully meets all assignment requirements. It provides a robust platform for analyzing complex scheduling scenarios, including multi-core environments and advanced server algorithms like Sporadic Server. The modular design allows for easy extension of future algorithms.