Embedded System Task Scheduler for Embedded Linux

Teresa Co
Columbia University
New York, USA
tc3499@columbia.edu

Handong He
Columbia University
New York, USA
hh3152@columbia.edu

Xiao Lu
Columbia University
New York, USA
xl3586@columbia.edu

Abstract

Our goal in this project is to create and put into use a straightforward embedded system task scheduler that operates on an embedded Linux platform. Using fundamental scheduling techniques including round-robin scheduling and priority-based scheduling, the system will mimic multiple separate activities. The order in which tasks are carried out and the duration of each task will be decided by the scheduler. Additionally, the system will gather data regarding task execution, such as waiting and execution times as well as task switching patterns.

Understanding how scheduling algorithms function in embedded systems and how they affect system behaviour is the aim of this study. Despite its simplicity, the suggested system will illustrate key ideas in embedded software architecture and operating systems. We intend to obtain hands-on experience with job scheduling and resource management in embedded computing systems through the design, implementation, and assessment of this system. Our work is also motivated by lightweight embedded scheduler designs such as RIOS, which show that small scheduler frameworks can still be useful for embedded learning and experimentation [1].

1 Introduction

Many contemporary technologies, including smart devices, medical equipment, and industrial control systems, rely on embedded systems. These systems frequently have to operate with constrained computational resources while handling multiple tasks concurrently. Because of this restriction, effective task management is crucial to the system's proper and dependable operation. One crucial element that aids in managing how various jobs share the CPU and system resources is a task scheduler.

Embedded systems are made to carry out particular tasks inside larger systems. Embedded devices frequently have lower processor, memory, and energy capacities than general-purpose computers. However, these systems often have to manage several tasks at once. A smart device might have to gather sensor data, process incoming data, update displays, and interact with other devices, for instance. To guarantee that the system functions well and reacts to events promptly, these tasks must be effectively managed.

One crucial method that enables several processes to share system resources is task scheduling. A scheduler in operating systems determines which process should run at a given time and how long it should run before another task is permitted to run. Performance characteristics may vary depending on the scheduling algorithm used. While some algorithms prioritize crucial work or seek to reduce waiting times, others seek to ensure task fairness. Prior work comparing Linux schedulers in embedded systems has shown that different schedulers can lead to different trade-offs in interactivity, fairness, and scalability [2].

In embedded systems, where resource limitations necessitate careful control of system behaviour, an understanding of task scheduling is particularly crucial. It is possible to see how various scheduling policies impact system performance by putting in place a basic scheduler. For instance, a priority-based scheduler can permit more crucial processes to execute more frequently, whereas a round-robin scheduler divides CPU time equally among tasks.

Designing and implementing a software-based task scheduler that illustrates these scheduling concepts is the aim of this project. Running on an embedded Linux platform, the scheduler will oversee multiple simulated jobs that mimic typical system operations.

The software implementation of scheduling algorithms in an embedded Linux environment is the main emphasis of this project. The study will examine scheduling behaviour and simulate tasks. It will not entail creating unique hardware schedulers or putting intricate real-time operating system capabilities into practice. Rather, the project's goal is to demonstrate basic scheduling ideas in embedded systems in an understandable and instructive manner.

2 System Design and Project Specification

2.1 System Context

The C programming language will be primarily used in the implementation of the suggested system, which will operate on an embedded Linux platform. A number of separate tasks that are typical of embedded systems will be simulated by the system. The chosen scheduling algorithm will determine the CPU time required for each activity.

The system architecture will include several main components:

- Task Manager
- Scheduler Module
- Execution Controller
- Monitoring and Data Collection Interface

Each component will perform a specific function to support task scheduling and system monitoring.

2.2 Task Creation and Management

Designing the task structure and developing simulated tasks constitute the initial phase of the project. Every assignment will be a straightforward program that carries out fundamental tasks like computations, waiting times, or simulated sensor processing.

Every task will have a number of characteristics, such as:

- Identifying the task
- Level of task priority
- State of execution (ready, running, or waiting)
- Statistics on execution time

A list or queue of open jobs will be kept up to date by the task manager. Additionally, task startup and state updates throughout execution will be handled by this module.

2.3 Scheduling Algorithm Implementation

The next job to be executed will be decided by the scheduler module. The next job will be chosen by the scheduler based on a scheduling policy after reviewing the list of available tasks.

There will be two scheduling algorithms used.

2.3.1 Round-Robin Scheduling. Round-robin scheduling allocates an equal amount of CPU time to each task. The scheduler places tasks in a circular queue and allotts a certain time slice to each task. The scheduler advances to the subsequent task in the queue whenever a task has completed its time slice.

Because each work is given CPU time, this technique is straightforward and guarantees fairness.

2.3.2 Priority-Based Scheduling. Every task is given a priority value in priority-based scheduling. Higher priority tasks are scheduled ahead of lesser priority chores. Important tasks can run more frequently using this strategy.

However, if higher-priority jobs continue to run, this approach can make lower-priority processes wait longer.

In order to compare the behaviour of these two scheduling algorithms, the system will enable switching between them. This comparison is meaningful because earlier work on Linux schedulers has shown that different scheduling policies can produce different behaviour in interactivity and fairness [2].

2.4 Task Execution and Context Switching

The actual task performance will be overseen by the execution controller. When the scheduler chooses a new task, the controller will mimic context shift between tasks.

The process of conserving the current task's state and restoring the state of another task so it can execute is known as context switching. Instead of implementing context switching at the operating system kernel level, this project will simulate it at the application level. This simpler design is appropriate for a lightweight educational or experimental scheduler [1].

When scheduling decisions are made, the execution controller will monitor the duration of each task and update task states.

2.5 Monitoring and Performance Analysis

The project will feature a monitoring component that logs scheduling statistics in order to better understand system behaviour.

The system will gather data like:

- Order of task execution
- Each task's CPU time
- Waiting period prior to execution
- The quantity of context changes

A straightforward interface, such terminal output or log files, will be used to display these facts. It will be feasible to examine how various scheduling strategies impact system performance by looking at these outcomes.

3 Plan, Milestones, and Risks

3.1 Project Timeline

The project will be completed in several stages.

M1: Literature Review and System Design. In this phase, the system architecture will be created and current scheduling algorithms will be examined. We will define the fundamental framework for tasks and scheduling modules.

M2: Task Management Implementation. The task management module and task generation mechanism will be put into use. To mimic system activity, a number of test tasks will be developed.

M3: Scheduler Implementation. The system will incorporate both the priority-based scheduler and the round-robin scheduler.

M4: Monitoring and Data Collection. The monitoring module will be developed to collect and display execution statistics.

M5: Evaluation and Documentation. We will test and compare several scheduling options. The outcomes will be examined and recorded.

3.2 Key Challenges

The execution of this project may provide a number of difficulties.

First, accurately simulating task switching behaviour while preserving consistent task states could be challenging. It will be necessary to manage task data structures carefully.

Second, careful timing techniques may be needed to ensure accurate task execution and waiting time measurements.

Lastly, even when several jobs vie for CPU resources, the project must make sure that the scheduling algorithms function properly. Prior research has shown that scheduler design can involve balancing multiple goals, including responsiveness, fairness, and timing requirements [2, 4]. In some embedded settings, scheduling may also be connected to energy-aware optimization [3].

Systematic testing and incremental development will be used to overcome these issues.

4 Expected Outcomes

The finished system will show how scheduling several jobs in an embedded context is possible. The project will shed light on how scheduling choices affect system responsiveness and performance by contrasting various scheduling approaches.

Despite its simplicity, the system will demonstrate important concepts in embedded software design and operating systems. The project will give participants hands-on experience with task management and scheduling techniques in embedded computing systems.

References

[1] Bailey Miller, Frank Vahid, and Tony Givargis. RIOS: a lightweight task scheduler for embedded systems. In *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*, 2012.

[2] Yigui Luo and Bolin Wu. A comparison on interactivity of three Linux schedulers in embedded system. In *Proceedings of 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*. IEEE, 2011.

[3] Vishnu Swaminathan and Krishnendu Chakrabarty. Real-time task scheduling for energy-aware embedded systems. *Journal of the Franklin Institute*, 338(6):729–750, 2001.

[4] M. V. Panduranga Rao et al. Development of scheduler for real time and embedded system domain. In *22nd International Conference on Advanced Information Networking and Applications - Workshops (AINA Workshops 2008)*. IEEE, 2008.

[5] P. Sivakumar et al. Real-time task scheduling for distributed embedded system using MATLAB toolboxes. *Indian Journal of Science and Technology*, 8(15):1–7, 2015.

[6] Doug Abbott. *Linux for Embedded and Real-Time Applications*. Newnes, 2017.