INFO-F-404
Real-Time Operating Systems

Multiprocessors Scheduling Project

Authors :

FAEK Ilias - 000573067

LAAKEL Ousama - 000516197

Academic Year 2024 - 2025

Contents

1 Introduction

This project aims to:

1. Implement and simulate multiple variants of the EDF algorithm in both single-processor and multi-processor environments.
2. Analyze and assess the schedulability of a set of periodic or sporadic tasks using the following approaches:
   - Local-EDF for single-processor systems.
   - Global-EDF for multi-processor systems.
   - EDF($\hat{(}$k) for multi-processor systems with stricter priority constraints.
3. Validate the correctness and performance of the algorithms using simulators.

This report details the implementation of scheduling algorithms, the simulation approach adopted, and the challenges encountered in managing jobs on single or multiple processors. It focuses on the efficiency of scheduling policies to ensure tasks meet their deadlines, even in complex and highly loaded systems.

The developed tool enables the analysis of whether a task set is schedulable under EDF and provides a practical framework for evaluating real-world scenarios in real-time systems.

2 Methodology

2.1 Algorithm Implementation

2.1.1 Simulation

Simulation pour Local-EDF The `simulate_local` function executes the local EDF scheduling on a single processor.

Explanation

- At each time t, the function checks if new tasks are released (`release_jobs`) and whether deadlines are met.

- The EDF scheduler selects the task with the closest deadline to be executed.
- If a task is completed, it is removed from the queue.

Algorithm 1 Simulate_Local(taskset, feasibility_interval)

```
1: Input: TaskSet $taskset$, Interval $[t_{min}, t_{max}]$
2: Output: 0 (Schedulable) or 2 (Non-schedulable)
3: Queue $\leftarrow \emptyset$                                   ▷ Initialize the job queue
4: for $t \in [t_{min}, t_{max}]$ do
5:     Queue $\leftarrow$ Queue $+taskset.release\_jobs(t)$
6:     for Job $j \in$ Queue do
7:         if $j.deadline\_missed(t)$ then return 2   ▷ Non-schedulable: deadline missed
8:         end if
9:     end for
10:    $elected\_job \leftarrow edf\_priority(Queue)$
11:    if $elected\_job \neq$ None then
12:        $elected\_job.schedule(1)$
13:        if $elected\_job.is\_complete()$ then
14:            Remove $elected\_job$ from Queue
15:        end if
16:    end if
17: end for
18: return 0                                   ▷ Schedulable: all tasks meet deadlines
```

Simulation pour Global-EDF The `simulate_global` function implements global EDF scheduling on m processors.

Explanation

- Each processor can execute one task at any given time.
- Tasks with the closest deadlines are selected and assigned to available processors.
- The algorithm is compatible with EDF(̂k): for $k > 1$, a modified version of the scheduler is used to handle stricter constraints.

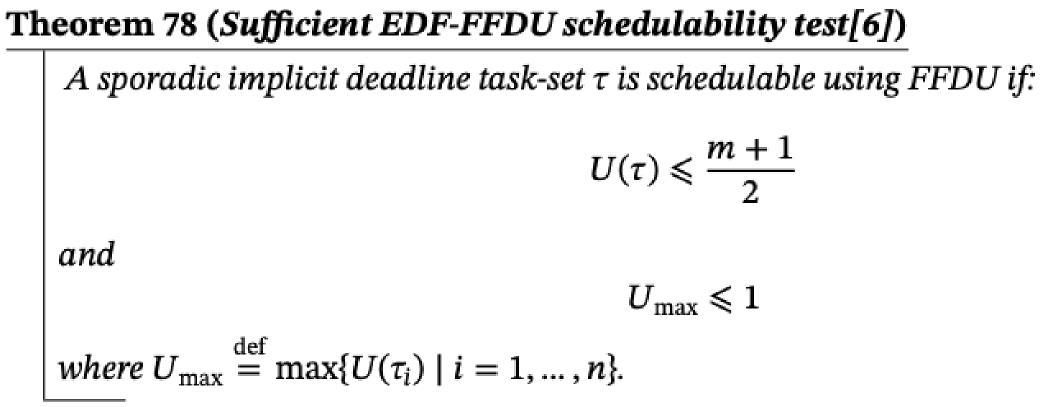
2.2 Schedulability Analysis

For EDF, $U(T) \leq 1$ is a necessary condition for uniprocessor schedulability. For each algorithm, we evaluated schedulability using additional schedulability tests. For Partitioned EDF, a sufficient condition exists only in the case of First Fit (FF) Decreasing Utilization (DU).

Algorithm 2 Simulate_Global(taskset, m, k, scheduler, feasibility_interval)

```
1: Input: TaskSet $taskset$, Processors $m$, Scheduler $scheduler$, Interval $[t_{min}, t_{max}]$
2: Queue $\leftarrow \emptyset$, Processor_Load $\leftarrow [None] \times m$
3: if $scheduler = k$ and $k \neq 1$ then
4:     Run EDF_k Scheduler
5: end if
6: for $t \in [t_{min}, t_{max}]$ do
7:     Queue $\leftarrow$ Queue $+taskset.release\_jobs(t)$
8:     for Job $j \in$ Queue do
9:         if $j.deadline\_missed(t)$ then return 2    ▷ Non-schedulable: deadline missed
10:        end if
11:    end for
12:    for $i \in [1, m]$ do
13:        if Processor_Load[$i$] $\neq None$ and Processor_Load[$i$].is_complete() then
14:            Remove Processor_Load[$i$] from Queue
15:            Processor_Load[$i$] $\leftarrow None$
16:        end if
17:    end for
18:    $elected\_jobs \leftarrow global\_edf\_scheduler(Queue, m)$
19:    for Job $j \in elected\_jobs$ do
20:        for $i \in [1, m]$ do
21:            if Processor_Load[$i$] = None or $j.deadline < Processor\_Load[i].deadline$ then
22:                Processor_Load[$i$] $\leftarrow j$
23:                $j.schedule(1)$
24:                break
25:            end if
26:        end for
27:    end for
28: end for
29: return 0                                        ▷ Schedulable: all deadlines met
```

2.2.1 Partitioned EDF: First Fit Decreasing Utilisation



Theorem 78 (*Sufficient EDF-FFDU schedulability test[6]*)

A sporadic implicit deadline task-set τ is schedulable using FFDU if:

$$U(\tau) \leqslant \frac{m+1}{2}$$

and

$$U_{\max} \leqslant 1$$

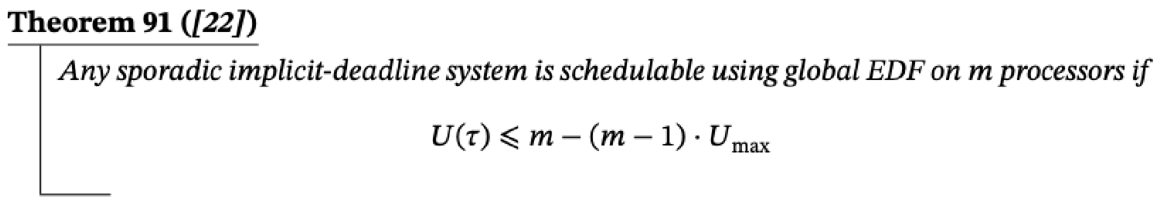
where $U_{\max} \overset{\text{def}}{=} \max\{U(\tau_i) \mid i = 1, \ldots, n\}$.

Figure 1: Theorem 78

In EDF-FFDU scheduling, a sufficient test is possible for determining schedulability. The idea of First Fit Decreasing Utilisation (FFDU) is to assign tasks (ordered by decreasing utilisation factor) by the use of first-fit.

2.2.2 Global EDF scheduling

A natural generalisation of EDF upon multiprocessors is global EDF for which a sufficient schedulability condition is known.



Theorem 91 (*[22]*)

Any sporadic implicit-deadline system is schedulable using global EDF on m processors if

$$U(\tau) \leqslant m - (m-1) \cdot U_{\max}$$

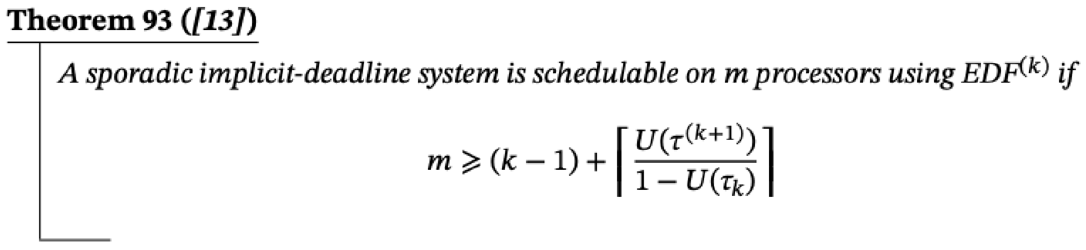
Figure 2: Theorem 91

2.2.3 EDF(k) scheduling

For simplicity, we assume that $U(\tau_1) \geq U(\tau_2) \geq \cdots \geq U(\tau_n)$. $\tau^{(i)}$ is used to denote the set of the $(n-i+1)$ tasks with the lowest utilisation factor of τ: $\tau^{(i)} = \{\tau_i, \tau_{i+1}, \ldots, \tau_n\}$.

We can derive from Theorem 91 that: The EDF(k) schedulability test (for k determined) is:

$$m \geqslant \left\lceil \frac{U(\tau) - U(\tau_1)}{1 - U(\tau_1)} \right\rceil$$


Theorem 93 (*[13]*)

A sporadic implicit-deadline system is schedulable on m processors using EDF$^{(k)}$ if

$$m \geqslant (k-1) + \left\lceil \frac{U(\tau^{(k+1)})}{1 - U(\tau_k)} \right\rceil$$

Figure 3: Theorem 93

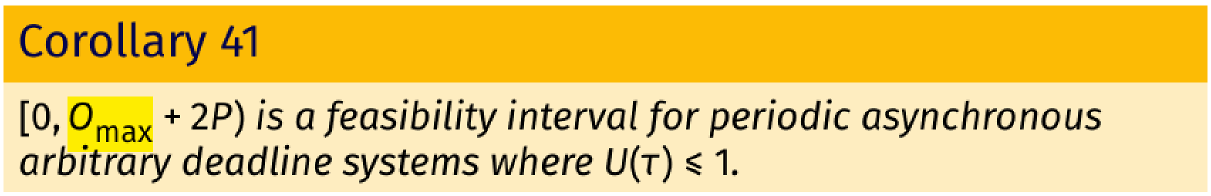
2.2.4 Feasibility interval

The feasibility interval is the time interval during which scheduling is simulated or analyzed to determine whether a task system is schedulable. **Partitioned EDF** The general formula is $[0, O_{\max}+2P]$. The hyperperiod is defined as the least common multiple (LCM) of the periods of the tasks in a periodic system. When simulating or analyzing a schedule, it is sufficient to observe an interval equal to the hyperperiod to determine if the system is schedulable. The offset of a task corresponds to the initial release time of that task. The maximum offset is the largest offset among all the tasks in a system.

When performing a feasibility analysis, it is necessary to start the observation from $O_{\max}$, as this is the moment when all tasks have been released. In the case of the Partitioned EDF algorithm, tasks are assigned to processors independently. Each core (or processor) then operates as an autonomous uniprocessor system.

For Partitioned EDF, the interval $[0, O_{\max} + 2P]$ is sufficient because each processor executes its tasks independently. This simplifies the analysis, as the observation can start at $t = 0$ without requiring synchronization with the offsets of other cores. In contrast, for global algorithms such as Global EDF or EDF(k), it is more appropriate to start the interval at $O_{\max}$, as these algorithms account for interactions between tasks across multiple processors.

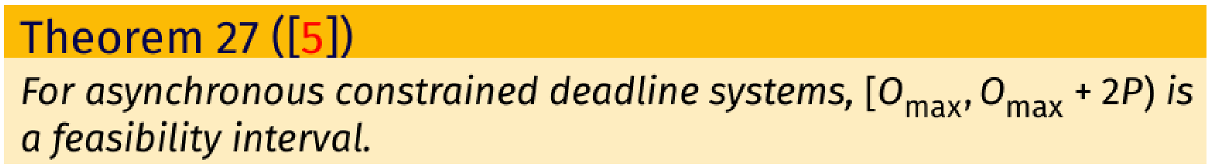
Global EDF & EDF(k)



Corollary 41

[0, O_{max} + 2P) is a feasibility interval for periodic asynchronous arbitrary deadline systems where U(τ) ≤ 1.

Figure 4: Partitioned EDF interval



Theorem 27 ([5])

For asynchronous constrained deadline systems, [O_{max}, O_{max} + 2P) is a feasibility interval.

Figure 5: Global EDF & edf(k) interval

2.3 Parallelization

2.3.1 What Can Be Parallelized?

The scheduling problem involves various operations, but not all are parallelizable:

- **Parallelizable:** Simulations for each core in partitioned scheduling are independent since tasks assigned to one core do not affect others. This independence allows the use of parallelism to simulate each core's schedulability simultaneously.
- **Non-Parallelizable:** Global EDF scheduling and EDF(k) require task dependencies and interactions across cores, making them less suited for straightforward parallelization. These methods need coordination to evaluate task interactions, which introduces overhead if parallelized.

2.3.2 Methodology and Criteria for Parallelization

We could not implement the parallelized simulation code, but we assumed that it's only a function of local schedulability in the `schedule_partitioned_edf` function. The criteria were:

- Independence of computations: Each core's simulation is independent, ensuring no need for inter-process communication.
- High computational cost: Simulating each core's schedulability is computationally

expensive, making it a candidate for performance improvement through parallelization.

- Feasibility of implementation: Using Python's `multiprocessing` module, we implemented parallelism with minimal modifications to the original code.

2.3.3 Role of Workers in Parallelization

Workers are the units of parallel execution that allow the program to distribute computations across available CPU cores. Their role is crucial in determining the efficiency of the parallelization:

- **Task Distribution:** Each worker is assigned a subset of tasks to simulate. In this implementation, workers independently simulate schedulability for cores.
- **Scalability:** Increasing the number of workers can reduce execution time by utilizing more CPU resources. However, the optimal number of workers is typically bounded by the number of physical CPU cores to avoid overhead from excessive context switching.
- **Overhead Management:** While more workers increase parallelism, they also introduce overhead related to process creation, data communication, and synchronization. This is why performance gains diminish as the number of workers exceeds the number of available CPU cores.

3 Conclusion

In this report, we analyzed the feasibility intervals for both Partitioned and Global scheduling algorithms under EDF and its variants. Partitioned EDF benefits from its simplicity, as each processor operates independently, allowing the analysis to start at $t = 0$. This independence reduces computational complexity and makes it a practical choice for systems with loosely coupled tasks.

In contrast, Global EDF and EDF(k) account for task interactions across processors, requiring the analysis to begin at $O_{\max}$ to capture inter-core dependencies.

While these algorithms offer more flexibility in resource allocation, they introduce additional complexity due to their global nature.

The study highlights the trade-offs between simplicity and flexibility in real-time scheduling, providing a framework for selecting the appropriate algorithm based on system requirements.