iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 Issue: II Month of publication: February 2025

DOI: https://doi.org/10.22214/ijraset.2025.67092

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

Self-Learning Earliest Deadline First Scheduler in Linux

P. Balakrishnan[1], Rajesh M[2], Rajesh R[3]
[1]NPOL/ DRDO, Cochin University of Science & Technologies (CUSAT), Kochi, Kerala, 682021, India
[2]National Institute of Electronics & Information Technology (NIELIT), Calicut, Kerala, 673601, India
[3]NPOL, DRDO, Kochi, Kerala, 682021, India

Abstract: This paper explains a novel adaptive scheduling policy named as "Self-Learning Earliest Deadline First" (SL-EDF) in Linux for real-time embedded applications. This scheduling policy eliminates the major limitation of SCHED_DEADLINE, the existing EDF policy in Linux. In Linux, scheduling policies do the task prioritization and assignment of tasks to a free processor in any real-time application. The proposed scheduler is a major improvement in terms of core utilization in multicore processors and also eliminates deadline misses for all tasks. With the SL-EDF policy, the core utilization is achievable up to the maximum value of "ONE". In this scheduling SL-EDF scheduler, any new task is assigned to the free core after ensuring that the core utilization will not exceed ONE. This paper elaborates the proposed design and implementation of the SL-EDF scheduler and improvement over existing EDF for better performance in multicore, real-time systems using Linux as Operating System.
Keywords: Core-utilization, Hyper-period, Scheduling policy, Self-Learning

I. INTRODUCTION

The objective of the research is to develop an adaptive scheduling policy for real-time applications on multicore processors eliminating the shortcomings of the existing SCHED_DEADLINE scheduling policy in Linux for real-time applications.

With the availability of embedded multicore processors, real-time systems are implemented using them to reduce power consumption and to make systems compact. The biggest challenge in using multicore processors is in the development of real-time system software. Linux operating system being available as open source is the most preferred OS by the application developers. Linux kernel has various scheduling policies that can be selected by the developer in the application program. The main function of any scheduler is to prioritize and allocate the tasks/threads of the application program. Effective utilization of the computational power of all the processing cores of a multicore processor and meeting the critical deadlines poses a great challenge in scheduling the tasks of any real-time application.

The objective of the real-time scheduling policies in Linux is to run the tasks of any real-time embedded system in a predictable manner. SCHED_DEADLINE, the Earliest Deadline First (EDF) is a real-time scheduling algorithm available in Linux kernel to prioritize the tasks based on their deadlines. As per this policy, at any point of time, the task having the smallest deadline will be on the top of the ready queue. EDF dynamically assigns priorities to tasks based on their deadlines, with the highest priority given to tasks with the smallest deadline. The EDF scheduling policy in the Linux kernel presents several challenges that impact its performance and usability in real-time systems

II. LITERATURE REVIEW

Scheduling policies in Linux prioritize the various tasks/threads and allocate to the free cores of the multicore processor. [1] The most suited real time policy in Linux to meet the critical deadlines is SCHED_DEADLINE (Earliest Deadline First) but is not generally used due to various limitations. While using this policy in an application software, the task parameters like Worst-Case Execution Time (WCET)(C_i), Deadline(D_i) and Period(T_i) are initialized. WCET initialized in the application program is the main parameter used by SCHED_DEADLINE scheduler for allocating the tasks to any free cores [2]. The parameter WCET varies depending on the co-running tasks on other cores of the multicore processor and hence it affects the task allocation at any point of time [3,4]. This paper discusses an adaptive EDF policy, an extension/ modification of the existing EDF.

The Earliest Deadline First (EDF) scheduler, SCHED_DEADLINE, in Linux has been widely studied for its strengths and limitations in real-time systems. Several studies have highlighted its shortcomings, particularly concerning scalability, overhead, and deadline guarantees.

EDF scheduling is used by the application developer in real-time systems, where tasks are prioritized by their deadlines. [5] provide an extensive review of periodic scheduling in time-triggered, hard real-time systems. [6] focuses on periodic tasks, managing task predictability and meeting stringent timing requirements by adaptive EDF approaches, particularly in systems needing both time-triggered and event-driven scheduling. [7] examined the defectiveness of SCHED_DEADLINE with respect to tardiness and hard real-time guarantees. [8] Error in runtime fed to the scheduler can lead to either under-utilization of system resources or excessive contention among tasks. This makes it extremely challenging for application software developers to optimize the real-time applications effectively. The paper [8] discusses the necessity of maintaining high CPU utilization to reduce the energy consumption, which is a major requirement for mobile and embedded systems where battery life is very critical.

III. PROPOSED METHOD

The limitations of SCHED_DEADLINE in the implementation of real time systems is addressed in the proposed SL-EDF policy. The main limitation of SCHED_DEADLINE is that the application developer must provide the parameters “runtime”, “deadline” and “period” for all the tasks/threads. The run time or the execution time of any task/thread is normally estimated by the application developer by running the task/thread in isolation in any of the cores. But in a real situation when all the cores of the multicore processor are loaded with different tasks, the execution time will vary depending on the interference of the co-running tasks on other cores. In the modified scheduler, SL-EDF, the worst-case execution time WCET is calculated while the tasks of the application program are running on the multicore processor. In every occurrence of a task, a mechanism is provided to use the modified WCET before assigning the task to any core. But in the existing scheduler in Linux the value fixed by the application programmer will not be changed.

A. System Model

Consider a multicore processor with M identical cores (P_1, P_2, ...P_M) and an application program consisting of a task set Γ with N (N>>M) tasks/threads ie, Γ= (τ_1.........τ_N) with execution time (C_1......C_N) and period (T_1......T_N).

1) Core utilization

Consider any task set Γ= (τ_1.........τ_N) with period (T_1......T_N) and execution time (C_1......C_N). In multicore processors, core utilization of the mth core is defined as $U_m = \Sigma \frac{C_i}{T_i} \quad \forall i$

In any real-time embedded system, almost all the tasks will be periodic in nature and are repeated in every hyper-period. Hyper-period is the LCM of the period of all the task in real-time system.

2) Core utilization in a Hyper-period

The task is assigned to any core such that it meets the core utilization criteria, $U_m \leq 1$. In a real-time embedded application, utilization of each core in a hyper-period is estimated as $H_{Um} = (m_1 * C_1/T_1 + m_2 * C_2/T_2 + \ldots.)/H$ where m_i indicates the repetition of the ith task in any hyper-period H and C_i, the worst-case execution time (WCET) of that task. The core utilization in any hyper-period must be maintained below ONE, $H_{Um} < 1$ to ensure that the tasks don’t miss their deadlines.

3) Computation of Worst-case execution time

In any complex real-time embedded system, a typical program developed for a multicore processor will have multiple tasks/ threads with different execution time, release time, deadline and periodicity. Periodicity, release time and deadline depend on the design of the system, but execution time varies with the interference from the other co-running tasks. In any real-time embedded system running on a multicore processor, the pattern of the tasks assigned by the scheduler to each core in multiple hyper-periods is observed by means of a program. As expected, the tasks assigned to any core in a hyper-period are repetitive in nature.

A scheme to compute the WCET in real-time when the application program is running is elaborated below. Initially, to estimate the execution time of any task in isolated condition, the task is executed in any core with no other cores loaded with any other task. This value is used as the initial value for any task in the application program. In normal situation, when all cores are running, the execution time will depend on the tasks running on other cores. Hence the WCET set by the application developer will not be correct and will lead to throttling issues in the system, leading to deadline misses by the tasks. As the tasks of any real-time embedded system are periodic in nature, the execution time is measured in every run of the task/thread. The worst-case execution time is the maximum of the execution time of the different instances of any task in a hyper-period. The WCET is estimated in the hyper-period and is averaged over multiple hyper-periods to get a reasonable value. This averaged value C_i is used in the calculation of the core utilization as given in equation (2).

B. Scheduling scheme in SL-EDF

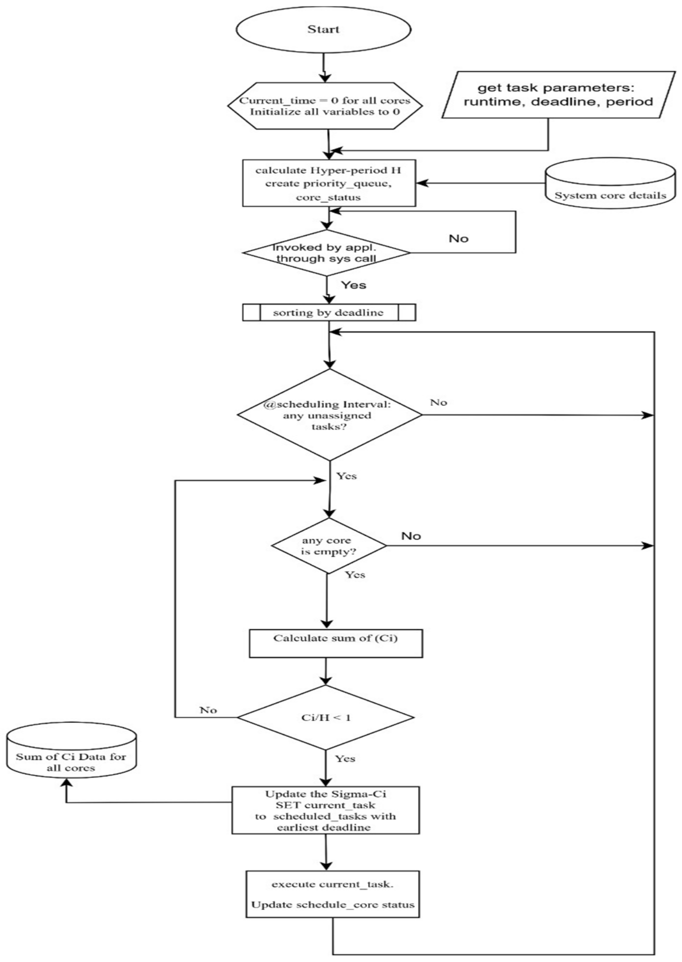
In each hyper-period, in the first instance of the task, the execution time, of the task is estimated. This is recorded in every instance of the task in the hyper-period. The highest value will give the WCET C_i of any task Ti.

In this scheduling policy, in any hyper-period, the summation of C_i/T_i of all the tasks scheduled to any core is calculated and the new task is allocated to any core if the summation $\sum Ci/Ti$ is < 1, after adding the new task.

The flow diagram for SL-EDF is given below in Fig 1. The flowchart illustrates the Self-Learning Earliest Deadline First (SL-EDF) scheduling mechanism, which dynamically updates the worst-case execution time to prevent CPU throttling and improve real-time scheduling efficiency. The process begins with the initialization phase, where the system sets Current_time = 0 for all cores and initializes necessary scheduling variables. Task parameters such as runtime, deadline, and period are obtained, and system core details are analyzed. The hyperperiod (H) computed as the Least Common Multiple (LCM) of all task periods. A priority queue is created, where tasks are sorted based on their earliest deadline, and each core status is updated to monitor available CPU resources.

Once the system is invoked through a system call, the scheduler checks for unassigned tasks and determines whether any CPU core is idle. If a core is available, the total execution time (sum(Ci)) is calculated, and the system evaluates whether the CPU utilization constraint (Ci / H < 1) is satisfied. If utilization exceeds the threshold, the scheduler waits for next core to become free. The process continues iteratively, ensuring that new execution time data from user-space tasks is incorporated, and scheduling adjustments are made dynamically in the next hyper period.

This approach effectively prevents CPU overload and deadline misses by continuously learning from execution patterns and adjusting scheduling parameters dynamically. By monitoring real-time execution time and modifying task runtime, SL-EDF eliminates the throttling issues that occur in traditional EDF scheduling, ensuring that real-time applications receive adequate CPU time without exceeding system capacity



No

Fig.1. Flow chart of SL-EDF

C. Implementation - Self-Learning EDF (SL-EDF) Scheduling in the Linux Kernel

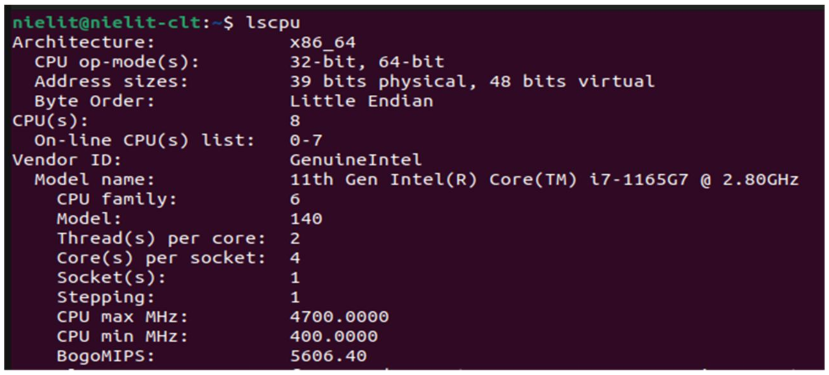
The Self-Learning Earliest Deadline First (SL-EDF) scheduling policy improves traditional EDF by dynamically adjusting task parameters based on the actual runtime of user-space real-time threads. Unlike conventional EDF, where the worst-case execution time (WCET) is predefined and may lead to CPU throttling if exceeded, SL-EDF periodically updates the execution time in the kernel. User-space threads measure their actual execution time and communicate this information to the kernel through a procfs interface. The kernel modifies the scheduling attributes accordingly.

At the core of SL-EDF, a kernel thread runs at fixed hyper period intervals, processing execution time updates and recomputing scheduling parameters. The kernel module utilizes the sched_setattr() system call to update task attributes under SCHED_DEADLINE, modifying the runtime dynamically. The SL-EDF framework consists of three primary components: user-space execution time reporting, kernel-space execution time management, and dynamic hyper period scheduling. User threads periodically write their observed execution times to procfs, allowing the kernel to collect and store this data. The kernel module reads these values and modifies the scheduling attributes of registered tasks. This approach makes EDF self-adaptive, reducing missed deadlines and ensures better CPU utilization while preventing priority inversion and overload scenarios common in traditional EDF scheduling. By continuously learning from execution behaviour of the tasks, SL-EDF enhances real-time scheduling, making it more efficient, reliable, and adaptive to dynamic system conditions.

D. Experimental setup and validation

This section explains the experimental platform and the results that validate our approach. An Intel embedded Xeon, 8 core processor was used to evaluate the performance of the SL-EDF policy. An Intel 8-core CPU running on Linux kernel version 5.19, is set up to test multi-threaded real-time applications. A total of 10 threads were created, each with variable computational load, execution time and period; all configured with the SCHED_DEADLINE policy. These threads were assigned to run across cores 1 to 7, isolated from core 0 where OS related tasks are scheduled. To maintain an isolated environment for these real-time threads, we shifted major Linux loads to core 0, thereby minimizing OS-related tasks on the other cores. This configuration allowed us to effectively test our scheduling policies by isolating Linux and background processes to core 0, focusing core 1 through 7 on deadline-sensitive tasks, and thereby optimizing performance in terms of deadline misses and load distribution across the testbed. The CPU configuration is listed in Fig 2

Fig 2 Testbed CPU specification



```
nielit@nielit-clt:~$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  8
  On-line CPU(s) list:   0-7
Vendor ID:               GenuineIntel
  Model name:            11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
    CPU family:          6
    Model:               140
    Thread(s) per core:  2
    Core(s) per socket:  4
    Socket(s):           1
    Stepping:            1
    CPU max MHz:         4700.0000
    CPU min MHz:         400.0000
    BogoMIPS:            5606.40
```

IV. RESULT AND DISCUSSIONS

We have considered 10 tasks/threads, named T0 to T9, with variable loads running on Linux OS, configured with the default scheduling policy SCHED_DEADLINE, as listed in Table 1. In this setup, the CPU is moderately loaded, with Linux running on core 0 and all threads running on cores 1 through 7. The threads execute over *n* iterations with *m* repetitions. We observed and recorded deadline misses across all iterations within the hyper-period. The experiments conducted confirm that the periodic tasks/threads of the application program are repeatedly assigned to the same core in the hyper-period. Under the existing EDF policy, the Ci/Ti values are not adequately accounted for in each iteration during scheduling, leading to a throttling and higher number of deadline misses, highly undesirable for real-time systems. SL-EDF checks whether each thread is schedulable on a core during core assignment and in every hyper-period, maintains the ΣCi/H ratio of each core to less than ONE. We have implemented a kernel module plugin that calculates and updates this value at every scheduling interval and provides input to the EDF scheduler, thus establishing a self-learning mechanism for EDF scheduling.

Ci - Execution Time of each task in ms ; SP - Scheduling Policy used;TS - Task set of 9 tasks; T - Period of each task in ms ; H – Hyper-period of the task set in ms ; R - Repetitions of each task in hyper-period; Dm - Deadline misses observed over hyper-period

Table 1. Thread parameter for a typical real-time application

TS	1	2	3	4	5	6	7	8	9	10
SP	SCHED_DEADLINE									
Ci	2	3	2	4	7	1.5	9	10	2	1.25
T	5	10	10	15	20	10	15	20	5	10
H	Hyper-period = LCM of Period = 60									
R	12	6	6	4	3	6	4	3	12	6
Dm	1	0	1	0	1	0	0	0	2	0

Table 2 shows the results obtained for any thread, example Thread 9, which is evaluated over *n i*terations within any hyper-period (12 iterations for thread 9).

H – Hyper-period ; I - Iteration No ; ID - Task ID ; Ci - Execution time of Thread T9 in ms ; W - WCET of T9

Table 2 shows the results obtained after implementing the SL-EDF. The results were compared with the readings of SCHED_DEADLINE policy. The results show that the SL-EDF has better schedulability compared to existing EDF policy in the Linux environment. The results clearly indicate that core utilization is always less than 100% in SL-EDF, whereas the utilization exceeds 100% in certain iterations in the case of SCHED_DEADLINE policy. When the core utilization exceeds 100% deadline misses will occur. Hence SL-EDF has better schedulability compared to the existing scheduling policy in Linux.

Table 2. Comparison of SCHED_DEADLINE Vs SL-EDF policy in a real time application

	ID	Ti (ms)	H_{Um} (%) (SCHED_DEADLINE)	H_{Um} (%) (SL-EDF)
	T0	5	101.3	93.4
	T1	10	78.98	77.09
	T2	10	102.24	98.88
	T3	15	71.56	70.08
	T4	20	100.08	96.55
	T5	10	69.03	66.88
	T6	15	82,34	80.78
	T7	20	95.34	94.88
	T8	5	100.78	97.99
	T9	10	85.97	84.76

V. CONCLUSION

The proposed SL-EDF scheduler improves existing EDF scheduling in terms of core utilisation and ensures zero deadline misses to real-time tasks of any real-time embedded application program. In the existing EDF policy, the execution time is initialised in the program, and the task to core assignment is based on this value. Here, the actual run time of the tasks is estimated while the tasks of the application program are executed. This enables the developer/user to port application programs to any hardware platform without modification. .

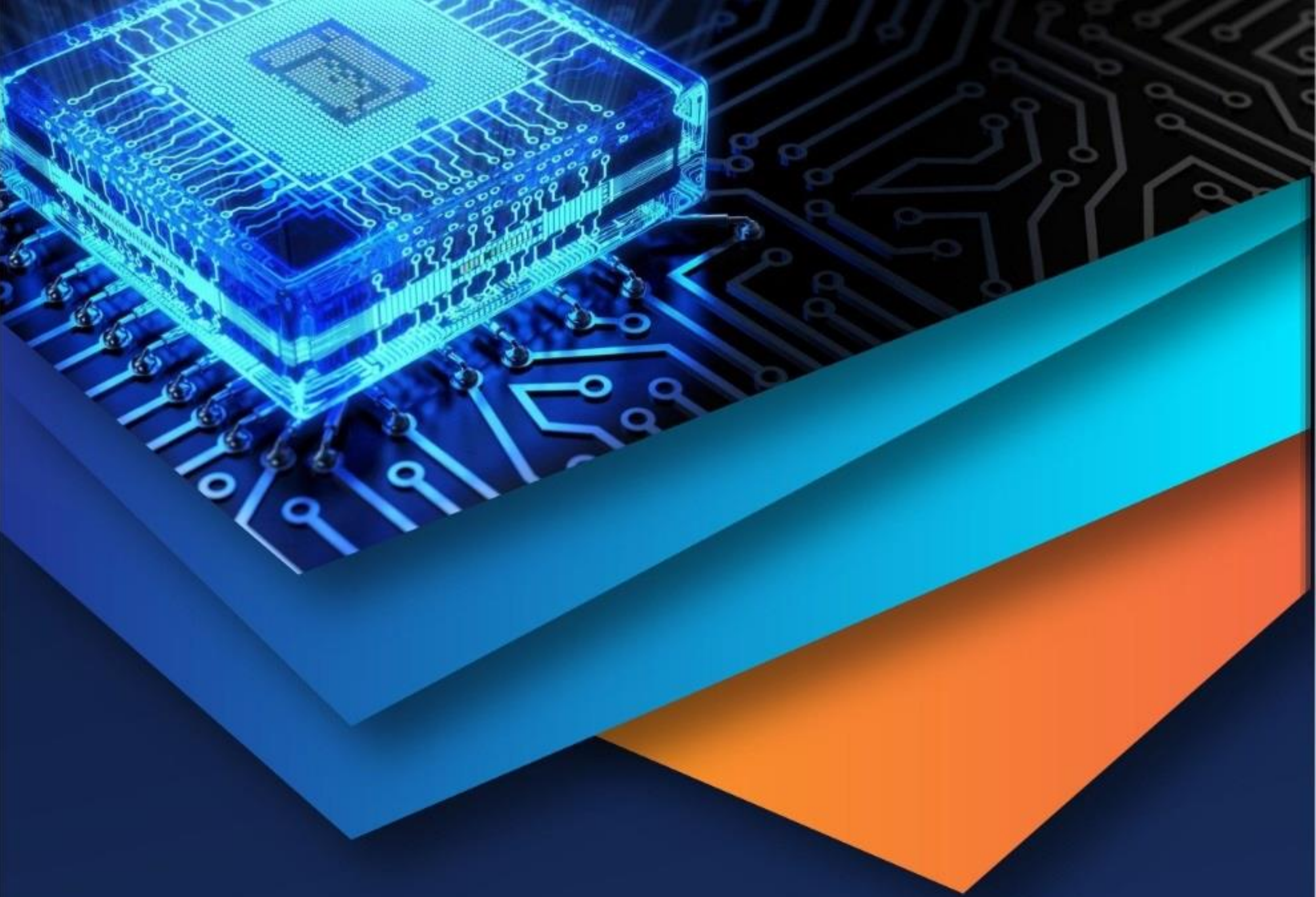
REFERENCES

[1] Chang S, Zhao X, Liu Z, Deng Q. Real-Time scheduling and analysis of parallel tasks on heterogeneous multi-cores. Journal of Systems Architecture [Internet]. 2020 May 1; 105:101704. Available from: https://doi.org/10.1016/j.sysarc.2019.101704

[2] Liang Y, Li H, Shen F, Xu Q, Hua S, Zhu S. Adaptive Multi-Core Real- Time Scheduling Based on Reinforcement Learning. 2024 IEEE 18th International Conference on Control & Automation (ICCA). 2024 Jun 18;148–53. Available in: https://doi.org/10.1109/ICCA62789.2024.10591927

[3] Muhuri PK, Rauniyar A, Nath R. On arrival scheduling of real-time precedence constrained tasks on multi-processor systems using genetic algorithm. Future Generation Computer Systems. 2019 Apr; 93:702–26. Available in: https://doi.org/10.1016/j.future.2018.10.013

[4] Tang S, Anderson JH, Luca Abeni. On the Defectiveness of SCHED_DEADLINE w.r.t. Tardiness and Affinities, and a Partial Fix. 2021 Apr 7;46–56. Available in: https://doi.org/10.1145/3453417.3453440
[5] Stevanato A, Cucinotta T, Luca Abeni, Bristot D. An Evaluation of Adaptive Partitioning of Real-Time Workloads on Linux. CINECA IRIS Institutional Research Information System (Sant'Anna School of Advanced Studies). 2021 Jun 1;53–61. Available in: https://doi.org/10.1109/isorc52013.2021.00018
[6] Liu X, Liu P, Yan X, Zou C, Xia R, Zhou H, et al. Energy Optimization and Fault Tolerance to Embedded System Based on Adaptive Heterogeneous Multi-Core Hardware Architecture. HAL (Le Centre pour la Communication Scientifique Directe). 2018 Jul 1;316–23. Available in: https://doi.org/10.1109/qrs-c.2018.00063
[7] Saez S, Vila J, Crespo A, Garcia A. A hardware scheduler for complex real-time systems. 2003 Jan 20; Available in: https://doi.org/10.1109/isie.1999.801754
[8] Minaeva A, Zdeněk Hanzálek. Survey on Periodic Scheduling for Time-triggered Hard Real-time Systems. ACM Computing Surveys. 2021 Mar 5;54(1):1–32. Available in: https://doi.org/10.1145/3431232
[9] Günzel M, Chen KH, Chen JJ. EDF-Like Scheduling for Self-Suspending Real-Time Tasks. arXiv (Cornell University). 2021 Jan 1; Available in: https://doi.org/10.48550/arxiv.2111.09725
[10] Vestal S. Techniques for Multiprocessor Global Schedulability Analysis. 2007 Dec 1; Available in : https://doi.org/10.1109/rtss.2007.35
[11] Tang S, Sergey Voronov, Anderson JH. Extending EDF for Soft Real-Time Scheduling on Unrelated Multiprocessors. 2021 Dec 1;253–65. Available in: https://doi.org/10.1109/rtss52674.2021.00032
[12] Lee S, Park S, Lee J. Improved Low Time-Complexity Schedulability Test for Non preemptive EDF on a Multiprocessor. IEEE Embedded Systems Letters. 2021 Dec 9;14(2):87–90. Available in: https://doi.org/10.1109/les.2021.3133901



10.22214/IJRASET

INDEX COPERNICUS

45.98

IMPACT FACTOR: 7.129

TOGETHER WE REACH THE GOAL

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)