Adaptive Real-Time Scheduler for Embedded Operating System

Arkajit Datta
BTech, School of Computer Science Engineering (SCOPE)
Vellore Institute of Technology (VIT), Vellore, India
arkajit.datta2019@vitstudent.ac.in

Shamith D Rao
BTech, School of Computer Science Engineering (SCOPE)
Vellore Institute of Technology (VIT), Vellore, India
shamithd.rao2019@vitstudent.ac.in

C.G Mohan
Associate Professor Grade-I
Vellore Institute of Technology (VIT), Vellore, India
Mohan.cg@vit.ac.in

Abstract

In CPU scheduling the classical problem of real-time and time-sharing operating systems is to schedule a sequence of jobs; the only given information, in this case, is the processing and completion time of the particular processes. All the previous scheduling algorithms focused on the system-level abstraction for the scheduling decision functions or the functions which implement them. In this research, a new scheduling algorithm, an Adaptive Multi-level Queue scheduling, will maximise the CPU utilisation and cater to the energy efficiency of the embedded system. This scheduling algorithm can adapt to the environment. The scheduling algorithm will be helpful in the embedded devices which are using a fixed battery. The scheduler combines various types of well-established scheduling algorithms like – Round-robin, Priority (Pre-emptive) and Multi-Level Queue feedback algorithms. The Adaptive feature incorporates a sleeping function in the scheduling algorithm which helps in the system energy optimisation**.**

Keywords
Embedded operating system; scheduling; round-robin; priority scheduling; multilevel feedback queue algorithm; and embedded systems.

1. Introduction
According to the well-established definition an embedded OS is well calibrated operating which is used to perform particular tasks for a device which is not a computer. The base-level job of an embedded system is to execute the code/programme which can do the specific task. As all the operating systems do, the embedded operating system also acts as a bridge between the hardware and software of a system. An OS is software that allows software abstraction and manages hardware resources. The embedded operating systems are used in the micro-controllers and the micro-processors for various tasks. These tasks include process control, industrial automation, multimedia, communication and command control. We come across embedded operating systems in our daily life, for example – Microwave, Refrigerator, Laser Printers, Calculators, Electronic Toys, Automatic LED lights and Automated Water Filters; these are all embedded systems using embedded operating systems.
Various types of RTOS –

1. *Hard Real-Time Systems*
2. *Soft Real-Time Systems*

The OS where all the computations need to be performed in a specific time is known as Hard RTOS. On the Contrary, the operating system where strict adherence to time is not guaranteed is known as the soft real-time operating system.

Scheduler being the most important component of an operating system provides developers with real-time management capabilities. Optimising the scheduler is mandatory in the operating systems to increase the device's CPU utilisation and throughput. There can be issues if we are not selecting the appropriate scheduling algorithm for the OS. The lag between the input and the output time should be considered undersized for the results to be accepted. Results delivered late can be regarded as to be useless and harmful. This research will devise a new scheduling algorithm, an Adaptive Real-Time Scheduler. Further, in the 2nd Section, the Literature Survey is being visited, the 3rd Section contains the contribution, problem statement, objective and the proposed framework, and Section-4 includes the conclusion.

1.1 Distinguishable Characteristics of Embedded OS

Embedded Operating systems are based on Read-Only Memory (ROM), it doesn't have Hard Disk or Floppy Drive. So, that's the reason embedded systems are of small size

The Responses in the Embedded Systems must be very fast as many devices control essential activities. Such as Traffic Lights, as soon as the signal is received, it must change traffic lights. This process must be done quickly and quite efficiently. In the embedded system, effective memory management and file management is required but not as sophisticated as the file management system and memory management systems in PCs and Desktops Operating System.

1.2 Features Of Embedded Systems

There are various features in embedded systems like multitasking where priority scheduling is predominant in embedded systems. We can get Quick Responses to External Interrupts. There are small kernels and fast context switches which are required in real-time operating systems. The embedded operating systems generally have a low-level system architecture therefore fast context switching and small kernels are required. The system also supports a real-time clock as an interface time interface and mechanism for the process of communication and synchronization.

2. Literature Survey

The round-robin scheduling protocols have various utilities other than particularly in operating systems, stochastic non-linear systems of neural networks are being created under the RR scheduling algorithm. The RR scheduling algorithm is considered effective in the mitigation of data congestion and saving energies zx mentioned by Ding et al. (2019). The scheduling algorithm of an Operating System is a significant resource and it decides the efficiency of the OS. The RR algorithm is the most used and trusted algorithm for scheduling. Efficiency is compared on the basis of the quantum and context switches. Amended Dynamic Round Robin (ADRR) is a new version of the RR algorithm with the primary goal of improving it using the active quantum-based time notion. The algorithm is compared on the basis of waiting time and turn-around time as mentioned in Shafi et al. (2020). Embedding Operating System must have an OS that is light weighted, energy-efficient, development-friendly and portable. Researchers have introduced an OS which is named RIOT, extensively for IoT and embedded systems. This operating system aims at energy efficiency and works on minimal memory, Baccelli et al. (2018). They cover the aspects crucial for a user/developer who will be using it as a product. They cover topics like hardware abstraction, kernel and software modularity, both conceptually and for various configurations. It also explains the procedural aspects of RIOT like the use of networking, system boot-up, power management, and timers in RIOT OS. Mostly, these characteristics of the operating system are aimed at assisting to improve code performance along with grounding the community support. There are however some aspects of RIOT that can potentially hinder the progress at the start. Irrespective of this drawback, RIOT has observed large-scale success that is on par with. In any way, the availability of such compact and powerful Operating systems would definitely revolutionize the field of the Internet of Things Baccelli et al. (2018). Round robin scheduling is also used over sensor networks where it is observed that it exploits to orchestrate the transmission order, under which it is observed that each node of the network broadcasts information to the neighbouring nodes. This particular algorithm helps in reducing communication costs as mentioned in Liu et al. (2019). A Real-Time scheduling policy, which is being extensively used in embedded and IoT systems has been proposed. It first studies and explains the real-time task scheduling algorithms, then went ahead with its comparison after which thitave a detailed conclusion with respect to the results. Mainly they made a comparison between the multiprocessor's real-time scheduling algorithms which also are classified into partitioned and global scheduling algorithms. The outcomes show that the partitioned algorithms are superior in execution compared to global scheduling algorithms with respect to makespan, waiting time, missed deadlines and task pre-emption. They also observed a higher performance in global scheduling algorithms when the number of tasks was raised but at the cost of an increased number of missed deadlines, Pathan (2018). DAG-Fluid is a planning calculation in light of liquid booking guan et al. (2020). Liquid planning depends on the way that the

errands are rescheduled after each ε of time. As ε tends to 0 the scheduler can switch the task status from running to pending and vice versa in an efficient manner making it look as though it runs continuously on a fraction of a processor. The paper demonstrated that the limit increase bound is 2−1/(m+1) which is the most reduced among all the current booking calculations for continuous equal DAG undertakings. Their trials likewise show that the DAG-liquid has beaten other cutting-edge calculations with regards to schedulable tests under exchanging condition/boundary settings. They additionally thought of an original methodology for building a timetable for non-liquid stages that displays similar attributes as DAG-liquid, guan et al. (2020).

Another age brilliant meter's innovative work foundation, then its overall programming and equipment design and the executive's unit equipment advancement stage, then, at that point, the development of plan climate and improvement apparatuses, as well as the essential capacities in view of the framework stage, lastly the product application improvement practice is proposed by Zhao et al. (2022). Makespan minimization problem in the dynamic job shop scheduling problem is along an adaptive scheduling technique is proposed where the directed acyclic graph is used to express the complicated precedence constraints across operations in jobs instead of a linear sequence as mentioned in Cao et al. (2019). A versatile booking calculation for the concurrence of ONUs with various tuning times in virtual PON, which is known as the multi-tuning-time ONU planning (MOS) calculation is proposed by Wang et al. (2019). Additionally, the reproduction demonstrates the way that the MOS calculation can really stay away from the additional line postpone brought about by ONUs' frequency tuning and lessen the misuse of transfer speed assets. The proposed scheduling algorithm in this paper tries to allocate CPU resources equitably among soft real-time jobs that may be starved in overloaded instances while focusing on the execution of high-priority heavy real-time jobs as its primary goal as mentioned in Malik et al. (2019). The best-worst multi-criteria decision-making technique (BWM) and the compromise ranking method are combined in this paper to present a novel adaptive methodology (VIKOR). The VIKOR technique is used to establish task priority as a decision-maker. The suggested method is tested through numerical tests and compared to current scheduling methods across a variety of performance measures in Rafieyan et al. (2020). For a wind energy change framework (WECS) in view of an injury rotor simultaneous generator, this examination offers a versatile and improved following technique utilizing a fluffy rationale step size regulator as stated in Gouabi et al. (2021). In a PSO-based task when the jobs are expected to be diverse, a scheduling strategy involving adaptive load adjusting is proposed where the Adaptive load adjusting method improves the proficiency of the exemplary PSO algorithm as mentioned in Ahmad et al. (2019). Self-Adaptive scheduling (SAS) algorithm for 5G base handset stations (BTSs) to upgrade energy proficiency, limit carbon impression and layout a self-supporting green cell network is presented in the examination by Dutta et al. (2018).

In the case of tasks that are generally diverse, this work provides a PSO-based task scheduling algorithm that uses a robust load balancing procedure. The suggested PSO-ALBA method uses robust load balancing to improve the performance of the regular PSO algorithm Ahmad et al. (2019). This research provides a robust genetic algorithm crossover mutation probability initial optimization approach (AGA). To make our procedure work, we utilize a double-coded chromosome. We contrast the superior versatile hereditary calculation with the versatile hereditary calculation (AGA) and the standard hereditary calculation utilizing tests (SGA) by Yiqiu et al. (2019). This study provides a job organising technique for diversified workloads in the cloud that is both efficient and effective. This will ensure that resources are used efficiently. We also offer a task scheduling strategy in a hybrid cloud based on BP neural networks to ensure that tasks are done within the user's specified timetable in Li et al. (2019). We proposed a double Q-learning (D-Q) technique in blend with the ongoing component of support figuring out how to work on the flexibility to natural changes by self-learning, like AI calculations, yet for the gathering position shop planning issue, to address the vulnerability of the creative climate in getting together workshops as mentioned in Wang et al. (2021). Cloud computing has gained a lot of importance in today's world. To optimise its use and ensure proper use of resources we aim to propose a task scheduling algorithm to manage workloads, reduce loss of time, and achieve accuracy and correctness as stated in Ibrahim et al. (2021). Many studies have concentrated on processor energy management. However, DPM for real-time periodic activities with shared resources via I/O device scheduling has received little attention. For shared resources, we address the problem of minimising I/O device energy consumption. We provide a fixed-priority scheduling approach for energy-conscious IO devices based on RM scheduling as mentioned in Zhang et al. (2020). In distributed computing, task scheduling is critical. It improves the system's performance by reducing overhead issues like communication delays and reducing overall execution time. For concurrent systems, a taxonomy of hierarchical classification is presented, and numerous task scheduling methods based on dependency are described by Tyagi et al. (2018). As a huge number of portable devices are developed, wireless sensor networks are becoming an emerging research topic. Because of this, operating systems (OS) can help to standardise the research and development of these systems and cut down on design time. This paper proposes a resource-constrained wireless device operating system scheduler that adapts job scheduling in various contexts as mentioned in Rodriguez et al. (2018).

3. Modules

3.1 Problem Description

In this study, our primary aim is to understand the crux of the policy mechanisms of various embedded OS scheduling algorithms in the domain of embedded operating systems and compare them to arrive at the best scheduling policy that can be applied with Adaptive nature.

3.2 Objectives

The objective of the study is to make sure that the scheduling algorithms are decent enough to fulfil the following criteria:

- Keeping the CPU busy 100% of the time with valuable work in order to maximise utilisation and efficiency.
- Maximizing the Throughput, which means reducing the false positives and false negatives, this would in turn make the system more reliable.
- Turn-Around Time (TAT): From the time of submission to completion.
- Waiting Time (WT): Minimizing this period.
- Response Time: Minimizing the response time for higher priority functions.
- Fairness: Ensuring that every process gets an equal share of the CPU runtime.
- Energy Efficiency: As most of the embedded systems are battery-operated, that's why the scheduling algorithm in the OS must make the system more energy efficient for a better user experience.
- Adaptive Nature: The scheduling algorithm must adapt its nature for better performance.

3.3 Proposed Framework

According to the literature survey and the problem statement, there have been specific ways in which the problem statement could be approached and solved.

- Identifying and optimising the best scheduling algorithm for the domain.
- Checking if the adaptive function (sleep function) can be added to the scheduling algorithm.
- Adding the Adaptive feature in the scheduling algorithm.
- Coding the proposed algorithm
- Checking the programme with various datasets.

3.4 Modules Description

Identifying and optimising the best scheduling algorithm for the domain:

Conducting a detailed study on different scheduling algorithms, the Multilevel Feedback Queue (MLFQ) has been identified as the most efficient scheduling algorithm for the embedded OS. This MLFQ scheduling algorithm considers both Priority (pre-emptive) and Round Robin scheduling. There will be different queues with different priorities; each queue will have a different time quantum. A process arriving will be sent to a queue according to its priority. Then we can find that there will be several other processes in the same queue with similar priorities. These processes in the queue can be scheduled as per the round-robin scheduling algorithm. The Adaptive Nature will be added to this algorithm, which will take the inputs such as the battery level of the system and then decide upon the sleeping period dynamically. The higher priority processes have to be done without any sleep function in between. Still, the lower priority processes could be delayed using the sleep function, which will increase the system's energy efficiency (Figure 1).

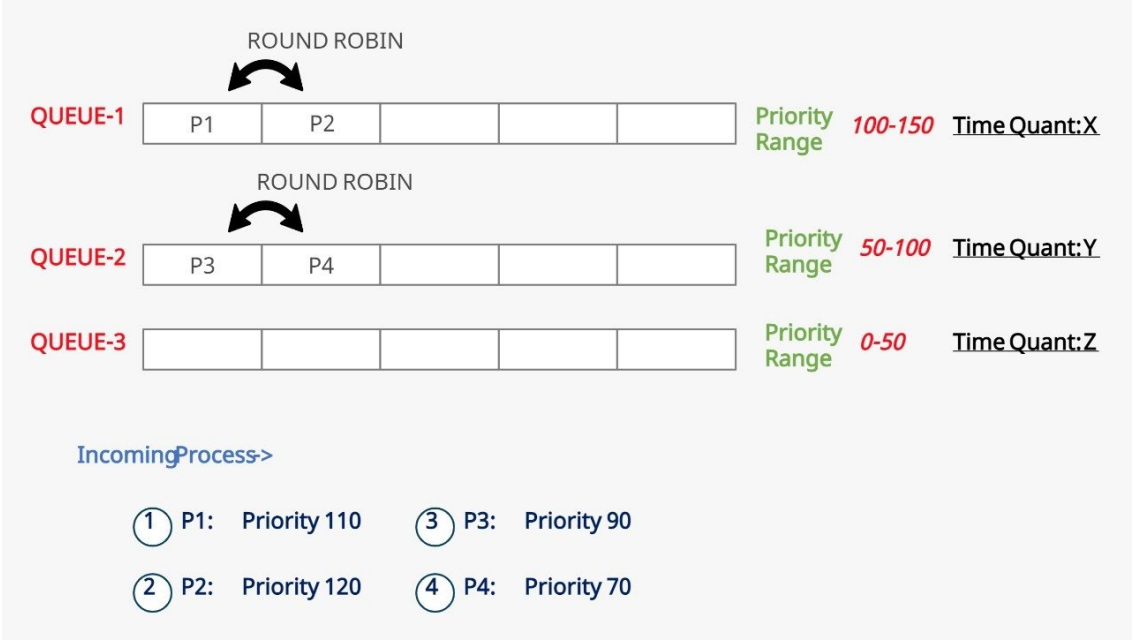

Figure 1. Multilevel Feedback Queue demonstration

Adding the Adaptive nature to the scheduling algorithm

As depicted in Figure 2, the Adaptive nature will be added to the Multilevel feedback Queue by incorporating a sleep function (sleep();), which will stop the ongoing process where ever it is and sleep the CPU. This process will be only done for low priority processes, which could be delayed without any commercial and significant losses.

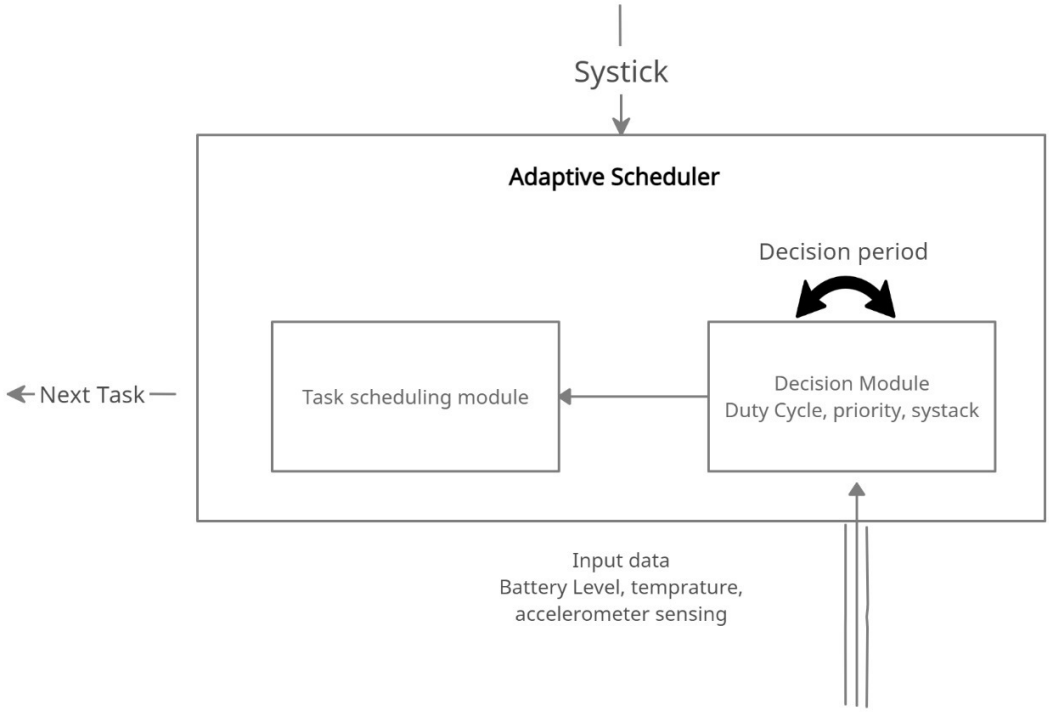

Figure 2. Adaptive scheduler flow diagram

In Figure 3, we can see that T3 being a higher priority job, the scheduler didn't apply the sleep() function there. Instead, use it when the same lower priority T1 and T2 processes were executed. The Duty Cycle and the Decision period are selected according to the number of functions available in the Queue.

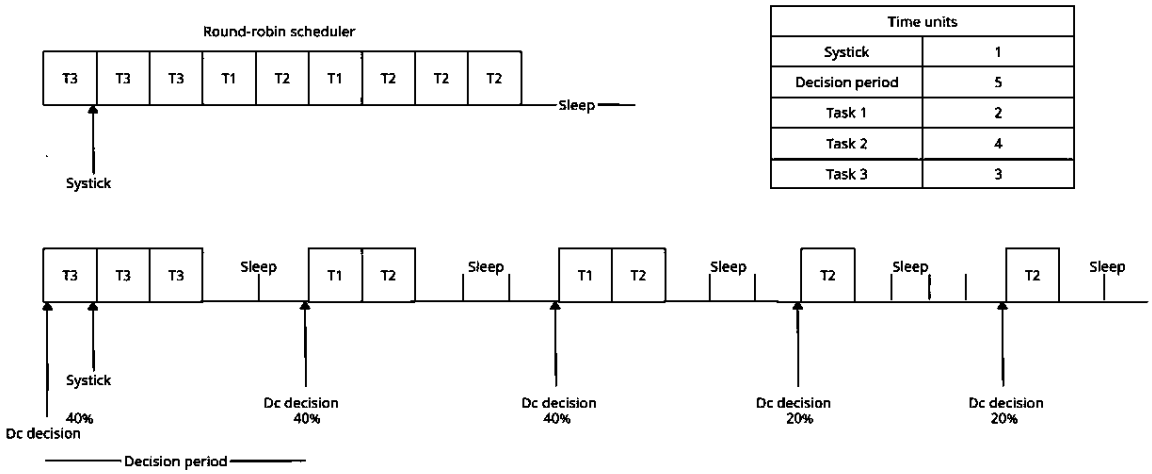

Figure 3. The change in the Gantt Chart After incorporating the Adaptive scheduler

4. Algorithm

The application of Multilevel-Feedback Queue (MLFQ), Priority and Round-robin scheduling is the base of this novel algorithm. The architecture of the algorithm contains various levels of Queues i.e., Queue1, Queue2, Queue3. Each queue schedules different priorities of jobs as depicted in Figure 1. After scheduling the appropriate jobs in their respective Queues, these jobs run in a round-robin fashion. As there are various levels of queues present, priority is given to the queues scheduling higher priority jobs (Figure 1). Also, each queue has its own Time-Quant essentially helping in giving more time to the higher priority jobs. The adaptive feature is added with the sleep function where the lower priority jobs can sleep for a certain time period. The main goal of the scheduler is to complete all the higher priority jobs at the earliest with saving the battery by sleeping the lower priority jobs. This particular feature would help in conserving the battery of the system as the CPU is not running at its hundred per cent potential always. Testing the algorithm, it was considered that there are 3 queues of different priorities with time-quants 3, 2 and 1 respectively. The sleeping time was set to 2 units. The code developed for the algorithm simulates the scheduling processes in a mathematical environment, which helps with insights into how the algorithm is reacting to various processes encountered. The Flow of the algorithm has been represented in Figure 4.

```
BEGIN

        time_quant_1 = 3;
        time_quant_2 = 2;
        time_quant_3 = 1;

        global_clock = 0;

        Structure of the Priority Queue{
                int pid //PROCESS ID
                int priority // PRIORITY OF THE JOB
                int time_of_arrival
                int process_burst_time
                Linked List is used to store all the jobs
                struct priorityqueue *left
                struct priorityqueue *right
        }

        The above-mentioned structure is used to create 3 different levels of
        queues.
        priorityqueue q1, q2, q3

        insert_in_queue(){
```

```
            This function inserts all the details into the Linked List of a
            particular priority queue
      }

      delete_node(){
            This function deletes a job after it is completed executing
      }

      check is_empty(){
            This fucntion checks if the queue is empty or not
      }

      queue_sort(){
            This function sorts the jobs in the queue with the arrival time of
            the job
      }

      sleep(int sleep_time){
            According to the input of the sleep_time, this function sleeps a
            job.
      }

      schedule(priorityqueue q1, time_quant, result){
            This is the main function that schedules and runs the jobs in a
            particular queue

            Sort the jobs with the help of the queue_sort()

            Then checks the clock and schedules the jobs arrived.

            If a job is completed it deletes the particular job with
            delete_node()

            If a lower priority job arrives it will incorporate the sleep
            function in it.
      }

      main(){
            Take the input of the jobs and store them in the respective queues

            schedule each and every queue with its priority and store the
            results

            Show the results
      }

END
```

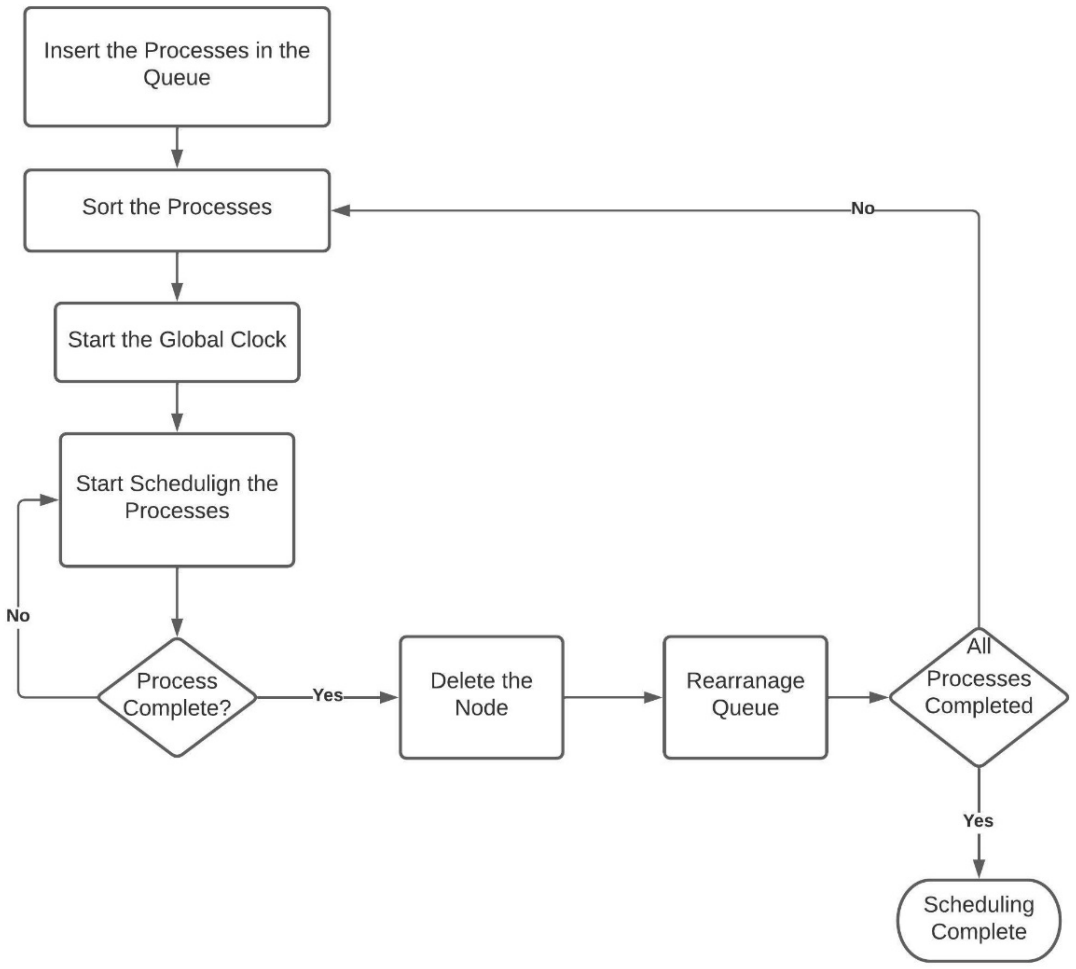

Figure 4. Flowchart representation of the proposed Algorithm

5. Results Discussion and Analysis

The Adaptive Multilevel Feedback Queue algorithm mentioned above is implemented using the C language. The Data-structure of Circular Queue using Doubly Linked List has been used for the same. Similarly, as stated in the Figure 1, three different queues with different priorities were made with attributes such as – Arrival time, Burst time, Priority and Process id.

Certain functions were made to Run the Algorithm. →

- **Insert ()** → for inserting the process in the Queue.
- **struct priority queue* delete_node(struct priority queue *node, struct priority queue *q)** → for deleting the process and freeing up the space when the process is done.
- **bool isempty(struct priorityqueue *q)** →This function will check whether the queue is empty or not, it will continuously compare the arrival time of the process with the global clock of the system.
- **void bubbleSort()** → This function will sort the processes according to the arrival time. If the input of the functions is not given according to the arrival time in sequential order, this function will do the sorting.
- **void sleep (int sleep_time)** →This method will be subjected to sleep the cpu whenever a lower priority job is scheduled. This function brings the adaptive nature to the Scheduling algorithm.
- **schedule()** → This main function schedules the process inside one queue in a Round-robin fashion and uses the delete function to delete the processes when it is done. After deleting the process, the results are saved in a structure called RESULT.
  The way Schedule function is called →
  r = schedule(q1,time_q1,r); //Highest Priority
  r = schedule(q2,time_q2,r); //Medium Priority
  r = schedule(q3,time_q3,r); //Least Priority
  Queues are scheduled based on the priority.

Our novel scheduling algorithm is tested and compared with one of the most used scheduling algorithm which is called Early Deadline First (EDF) for the embedded operating systems.
The same set of Few processes with similar arrival and burst time will be given to both the algorithms and then the CPU utilisation among both the algorithms will be compared. Also, the sleep time in the Adaptive Real-time algorithm will be shown in the Result Display part.

A Set of 6 processes are taken in which there are two processes each for each level of Queue.
Process 1 → arrival time = 0 ms, burst time = 10 ms, priority = 110 [High Priority]
Process 2 → arrival time = 1 ms, burst time = 15 ms, priority = 120 [High Priority]
Process 3 → arrival time = 2 ms, burst time = 20 ms, priority = 080 [Mid Priority]
Process 4 → arrival time = 3 ms, burst time = 10 ms, priority = 071 [Mid Priority]
Process 5 → arrival time = 4 ms, burst time = 25 ms, priority = 10 [Low Priority]
Process 6 → arrival time = 5 ms, burst time = 16 ms, priority = 15 [Low Priority]

The Sleep time has been statistically set to be = 2ms. In the main function initially, all the processes are being taken as inputs and then the logic is invoked where the process is subjected to various priority queues for further scheduling. Once all the processes reach their respective queues, they are being sorted according to their arrival time. After the sorting, the processes are ready for execution where a global clock maintains the flow of time and keeps a check on where which process is being executed. All the processes are scheduled using the round-robin (RR Algorithm) algorithm inside a queue hence there would be occurrences of the process waiting for getting executed, all this information is being logged as depicted in Table 1 which can be used for performance evaluation and comparison.

Table 1. Results for scheduling processes

PID	AT	BT	CT	WT
101	0	10	21	11
102	1	15	27	11
104	3	10	45	32
103	2	20	57	35
106	5	16	150	129
105	4	25	180	151
CPU Utilization – 1.850401				
Average Waiting Time - 61				
Average Completion Time (Average TAT) - 80				

The Programme was subjected to run for other test cases which are given below –
A Set of 6 processes are taken in which there are two processes each for each level of Queue.
Process 1 → arrival time = 0 ms, burst time = 20 ms, priority = 119 [High Priority]
Process 2 → arrival time = 2 ms, burst time = 15 ms, priority = 118 [High Priority]
Process 3 → arrival time = 4 ms, burst time = 20 ms, priority = 091 [Mid Priority]
Process 4 → arrival time = 5 ms, burst time = 11 ms, priority = 069 [Mid Priority]
Process 5 → arrival time = 9 ms, burst time = 25 ms, priority = 10 [Low Priority]
Process 6 → arrival time = 11 ms, burst time = 26 ms, priority = 15 [Low Priority]

The Results for the second test case are shown in Table 2

Table 2. Results for scheduling processes

PID	AT	BT	CT	WT
101	2	15	31	14
102	0	20	37	17
104	5	11	59	43
103	4	20	69	45
106	9	25	219	185
105	11	26	222	185
CPU Utilization – 1.731980				
Average Waiting Time - 81				

Average Completion Time (Average TAT) - 106

Here, PID represents Process ID, AT represents Arrival Time, BT represents Burst Time, CT represents Completion Time and WT represents Wating Time.
Here, we can observe the CPU utilisation is less in the Adaptive Real-Time scheduling algorithm compared to the EDF scheduling algorithm.
Adaptive Real-Time Scheduling – CPU UTILIZATION = 1.85
EDF – CPU UTILIZATION = 1.95

Comparing the Average WT and the Average TAT, we can observe that the EDF gives a better result.

EDF →
Average Waiting Time = 40
Average Turn Around Time = 57

Adaptive Real-Time Scheduling Algorithm →
Average Waiting Time = 61
Average Turnaround time = 80

This happens because of the Adaptive nature we have incorporated using the sleep function; this function delays the lower priority processes accordingly, giving us less CPU utilisation and helping to save battery. Figure 5 and Figure 6 compares the waiting and turnaround times of the scheduling algorithms.

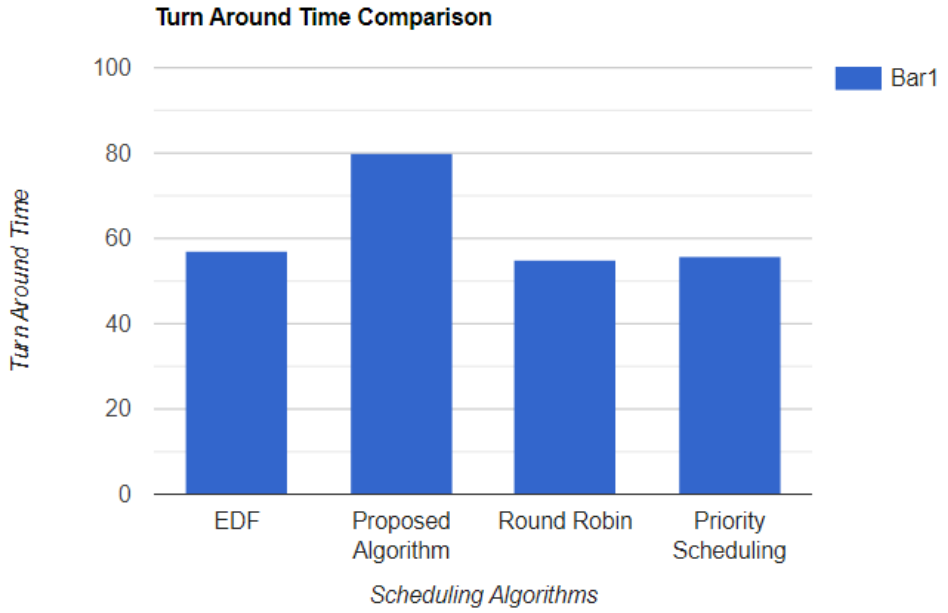

Figure 5. Average Turn Around Time (TAT) comparison between the scheduling algorithms

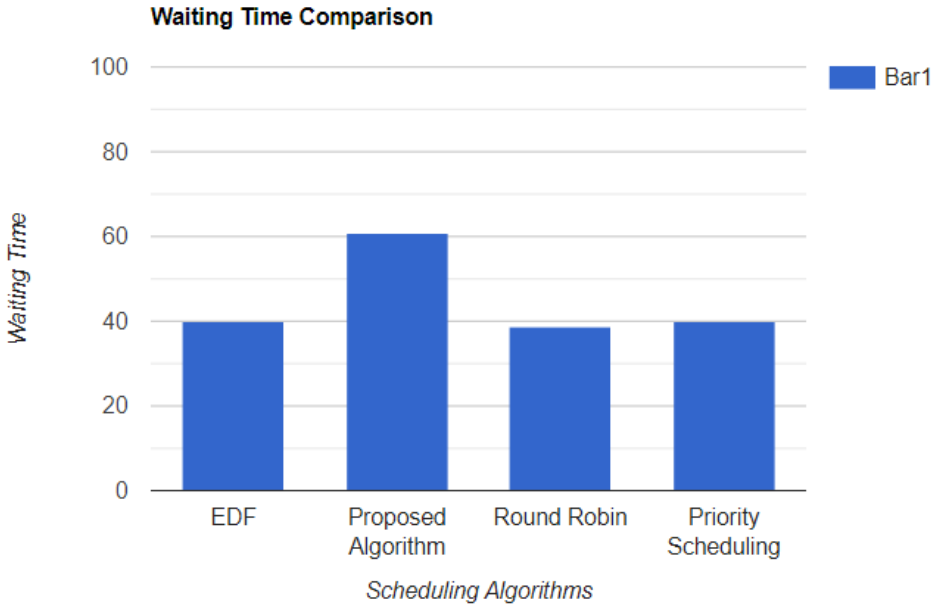

Figure 6. Average Waiting Time (WAT) comparison between the scheduling algorithm

6. Conclusion

The main contributions made in this study are - A detailed survey of the domain and various scheduling algorithms related to the field has been done. Proposing a novel adaptive scheduling algorithm which not only efficient in scheduling but also battery and power-efficient. Algorithms of the following implementations are stated under section 3.4. The implementation of the Algorithm developed was done and compared with the existing Algorithm, Early Deadline First. It has been observed that the CPU utilisation was less in the proposed algorithm of the Adaptive Real-time Scheduling algorithm compared to EDF. Also, the sleep function successfully helped delay the lower priority processes, which increased the average waiting time and average turnaround time.

The GitHub Link for the C Language Code - https://github.com/Arkajit-techie/ADAPTIVE-REAL-TIME-SCHEDULING-ALGORITHM/blob/master/adaptive_real_time_scheduler.c

References

Ahmad, M. O., and Khan, R. Z. Pso-based task scheduling algorithm using adaptive load balancing approach for cloud computing environment. *International Journal of Scientific & Technology Research*, vol. *8*(11), 2019.

Baccelli, E., Gündoğan, C., Hahm, O., Kietzmann, P., Lenders, M. S., Petersen, H., and Wählisch. RIOT: An open source operating system for low-end embedded devices in the IoT. *IEEE Internet of Things Journal*, vol. *5*(6), 4428-4440, 2018.

Cao, Z., Zhou, L., Hu, B., & Lin, C. An adaptive scheduling algorithm for dynamic jobs for dealing with the flexible job shop scheduling problem. *Business & Information Systems Engineering*, vol. *61*(3), 299-309, 2019.

Dutta, U. K., Razzaque, M. A., Al-Wadud, M. A., Islam, M. S., Hossain, M. S., & Gupta, B. B. Self-adaptive scheduling of base transceiver stations in green 5g networks. *IEEE Access*, vol. *6*, 7958-7969, 2018.

Ding, D. Z. Wang, Q. -L. Han and G. Wei, Neural-Network-Based Output-Feedback Control Under Round-Robin Scheduling Protocols, in IEEE Transactions on Cybernetics, vol. 49, no. 6, pp. 2372-2384, 2019, doi: 10.1109/TCYB.2018.2827037.

Fiad, A., Maaza, Z. M., & Bendoukha, H. Improved Version of Round Robin Scheduling Algorithm Based on Analytic Model. Int. J. Networked Distributed Comput., vol. 8(4), 195-202, 2022.

Guan, F., Qiao, J., & Han, DAG-fluid: A real-time scheduling algorithm for DAGs. *IEEE Transactions on Computers*, vol. *70*(3), pp. 471-482, 2020.

Gouabi, H., Hazzab, A., Habbab, M., Rezkallah, M., & Chandra, A. Experimental implementation of a novel scheduling algorithm for adaptive and modified P&O MPPT controller using fuzzy logic for WECS. *International Journal of Adaptive Control and Signal Processing*, vol. *35*(9), 1732-1753, 2021.

Ibrahim, I. M. Task scheduling algorithms in cloud computing: A review. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. *12*(4), 1041-1053, 2021.

Li, C., Jianhang, T., & Youlong, L. Hybrid cloud adaptive scheduling strategy for heterogeneous workloads. *Journal of Grid Computing*, vol. *17*(3), 419-446, 2019.

Liu, S., Wang, Z., Wei, G., & Li. Distributed set-membership filtering for multirate systems under the round-robin scheduling over sensor networks. IEEE transactions on cybernetics, vol. 50(5), 1910-1920, 2019.

Liu, D., Zhao, Y., Cai, X., Xu, B., & Qiu, T. Adaptive scheduling algorithm based on CPI and impact of tasks for multifunction radar. *IEEE Sensors Journal*, vol. *19*(23), 11205-11212, 2019.

Malik, S., Ahmad, S., Ullah, I., Park, D. H., & Kim, D. An adaptive emergency first intelligent scheduling algorithm for efficient task management and scheduling in hybrid of hard real-time and soft real-time embedded IoT systems. *Sustainability*, vol. *11*(8), 2192, 2019.

Pathan, R. M. Real-time scheduling algorithm for safety-critical systems on faulty multicore environments. *Real-Time Systems*, vol. *53*(1), pp. 45-81, 2018.

Rafieyan, E., Khorsand, R., & Ramezanpour, M. An adaptive scheduling approach based on integrated best-worst and VIKOR for cloud computing. *Computers & Industrial Engineering*, vol. *140*, 106272, 2020.

Rodriguez-Zurrunero, R., Utrilla, R., Romero, E., & Araujo, A. An adaptive scheduler for real-time Operating Systems to extend WSN nodes lifetime. *Wireless Communications and Mobile Computing*, 2018.

Shafi, U., Shah, M. A., Wahid, A., Abbasi, K., Javaid, Q., Asghar, M. N., & Haider. A novel amended dynamic round robin scheduling algorithm for timeshared systems. *Int. Arab J. Inf. Technol*., vol. 17(1), pp. 90-98, 2020.

Simaiya, S., Gautam, V., Lilhore, U. K., Garg, A., Ghosh, P., Trivedi, N. K., & Anand, A. EEPSA: Energy Efficiency Priority Scheduling Algorithm for Cloud Computing. In 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC) (pp. 1064-1069), 2021.

Tyagi, R., & Gupta, S. K. A survey on scheduling algorithms for parallel and distributed systems. In *Silicon Photonics & High Performance Computing* (pp. 51-64). Springer, Singapore, 2018.

Wang, X., Gan, C., & Tong, L. Adaptive scheduling algorithm for the coexistence of ONUs with different tuning time in virtual passive optical network. *IEEE Photonics Journal*, vol. *11*(5), 1-8, 2019.

Wang, H., Sarker, B. R., Li, J., & Li, J. Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning. *International Journal of Production Research*, vol. *59*(19), 5867-5883, 2021.

Yiqiu, F., Xia, X., & Junwei, G. Cloud computing task scheduling algorithm based on improved genetic algorithm. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)* (pp. 852-856), 2019.

Zhang, Y. W., Wang, C., & Liu, J. Energy aware fixed priority scheduling for real time sporadic task with task synchronization. Journal of Systems Architecture, vol. 83, 12-22, 2018.

Zhang, Y. Energy-aware fixed-priority scheduling for periodic tasks with shared resources and IO devices. International Journal of Embedded Systems, vol. 12(2), 166-176, 2020.

Zhao, X., Zhang, B., Yin, S., Xie, Q., Zhang, L., & Wu, H. Discussion on Application of Embedded Operating System in Dual-Core Smart Electric Meter. *Scientific Programming*, *2022*.