International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-11, September 2019

Performance of Dynamic Queue Based Minimal Deadline First Scheduling Algorithm In Multicore System Under Real Time And Non-Real Time Kernel Environment

M Shanmugsundaram, Kalpak Burgul, Kumar R, and Kittur H M

Abstract: *Some real-time systems that need to be associated with operating system services with a hard real-time system. Since these real-time systems that need to be extremely responsive to the outside world have no simple and low-cost operating system assistance. This paper deals with the application on a Linux-based operating system of the priority-based preemptive real-time scheduling algorithm that will suffice these firm applications in real-time. Typically, the algorithms regarded for these hard real-time systems are preemptive scheduling based on priorities. Based on the priority, by meeting the deadline, this algorithm can produce a feasible schedule for the dynamic tasks to be performed on the processor. It is feasible to schedule tasks on a processor as long as preemption is permitted and tasks do not compete for resources. In this scheduling algorithm, the task in the running queue that is waiting for the execution will be placed in the priority queue that is ready to execute in the available processor. This algorithm is deployed in the Linux kernel with the patch file and the kernel is built in the multi core system to execute an application.*

Keywords: *Real time scheduling; Preemptive priority; Real time systems; Kernel; MDF; DQMDF; Multicore; Raspberry Pi.*

I. INTRODUCTION

The processor is the essential component of an automatic data processing system and should be used more effectively. Once the demand for computing power increases, the possibility of missing the deadline, which is the disadvantage of task scheduling and load equalization, may be increased. More importantly, by not wasting any clock cycles, processor must be used at the highest rate. Such task scheduling should therefore conduct the execution of real-time tasks [1]. Real-time tasks are one in which accuracy depends not only on the precision of the result, but also on the time of completion of the task execution. These real-time tasks should eventually be scheduled in such a way as to satisfy the execution limitations at the deadline level [2].

It is very essential to periodically perform certain tasks in the processor. Examples include monitoring the external environment by periodically receiving the sensor information [3]. The regular execution of tasks therefore plays a significant role as they need to be processed within the limit so that the subsequent execution of tasks would not be influenced by the missing of the deadline leading to adverse effects in the response of the system. Most of these real-time systems require periodic monitoring analysis of the information. And in the real-time systems, scheduling these periodic tasks is very essential [4].

1.1 Real Time System

The external environment is monitored and controlled by real-time systems. The external environment information is provided to the microcontroller for processing as input and also to control the environment by meeting certain real-time limitations with the aid of actuators [5]. It is also possible to allow human link to this external universe with the suitable interface that can be used to provide the reference input. The terms used in the daily environment are usually referred to as systems because the accurate output is expected to be produced at the exact time as well [6]. Since it is essential for the system to respond at the right moment within the time limit, it is realized as a hard real-time system. Since the deadlines for these real-time systems can be predefined, the system's response is of profound importance in the desired way and at the right time. Therefore, in order to design a processor-based real-time system, it is essential for the processor to provide the real-time reaction within the desired deadline [7] for monitoring in the real-time environment. Failure to do so would lead to idiosyncratic and intolerable outcomes in environmental circumstances leading to dangerous impacts and inappropriate functioning. Taking the example of a rocket launch, if any of the rocket systems fail to react within the deadline, rocket launch failure and other catastrophic effects would inevitably result. The processor's functioning in real time is therefore essential [8]. Here, the processor must not only function properly within the time limit, but also prioritize the most important tasks in the real-time environment and preempt the task currently taking place with a high priority task. This scheduling is the most significant component of processing in real-time systems in order to prioritize and perform the significant tasks within the deadline and also to manage all remaining tasks with low priority by proper allocation of these tasks during the idle time of the processor. Hence, this leads to proper utilization of the processor while increasing the performance during the necessary time [9].

Manuscript published on 30 September 2019.
*****Correspondence Author(s)
M Shanmugsundaram, Vellore Institute of Technology, Vellore, Tamilnadu, India
(email: phdsundaram@gmail.com)
Kalpak Burgul, Vellore Institute of Technology, Vellore, Tamilnadu, India.
Kumar R, National Institute of Technology, Warangal, Telangana, India.
Kittur H M Vellore Institute of Technology, Vellore, Tamilnadu, India.

Real-time processing in the processor with multiple cores will improve the processor's computational capacity, efficiency, and parallelism. Therefore, multiple cores need a novel way to schedule and synchronize tasks between them in real-time processing. Multiple high-priority tasks can be performed concurrently on various processor cores over this scheduling [10]. Based on the dynamic task set of each process to be performed in the processor, the first scheduling algorithm is dynamically scheduled with dynamic queue-based minimum deadline. This schedule reduces the amount of deadline misses relative to static preemptive priority scheduling [11].

1.2 Implementation of Non-Real Time OS with Real Time Systems

In our daily routines, some generic real-time systems are extremely used. The systems must be sensitive to the deadline. This is one of the key characteristics that distinguish real-time systems from non-real time systems. So the programming and development firmware of real time systems need a real time operating system. Together with the processor, this operating system must have the essential computing capacity to supply the output within the critical time frame. Also, from other low-priority tasks, the real-time operating system must categorize the significant tasks to be carried out at first. This prioritization and scheduling is one of the significant features of the real-time operating system and in the non-real-time operating system this contrasting characteristic is absent. It is also not readily and conveniently affordable the operating system that is used to develop easy real-time systems. So it will take longer to develop this type of real-time systems, and even the cost may increase with the performance for trade-offs. Suppose this contrasting scheduling function is eradicated then these features can be readily applied in simple real-time, highly responsive systems. Consequently, implementing the real-time scheduling algorithm in a real-time operating system to develop real-time systems could improve feasibility and extend the future development area of highly accurate real-time systems [12].

II. SCHEDULING

The way the programming tasks are developed and analyzed will describe the system behavior. There is no adequate decision-making procedure for this behavior. Time and latency for processing will play a significant part in meeting the deadline. Every task in the task-set must be scheduled within the deadline without violating the optimal requirements associated with the feasible schedule. For real-time systems, preemptive priority scheduling is preferable [13]. The other real-time scheduling categories are discussed in detail in chapter 2.1.

2.1. Classification

Scheduling is the method used to perform certain tasks at a particular time. Real-time scheduling is not only about providing the right output, but also about the time required to produce results. The boundary within which systems need to respond in real time is known as a deadline [14]. Different scheduling methods that are categorized as Figure 1 are used to obtain optimal outcomes. These scheduling techniques are classified as preemptive and non-preemptive scheduling based on real-time or non-real-time implementation environments. Non-preemptive scheduling algorithms are preferable to non-real time system in which the execution of the next task begins only after the current task is completed. Examples are First Come First serve (FCFS) and Shorest Job First (SJF) scheduling algorithms. Recommended for real-time systems are the preemptive scheduling algorithms in which the highest priority task should be performed first. The prioritization is provided to the task on the basis of certain optimal criteria such as minimal deadline. Round robin scheduling is based on a clock tick that preempts tasks over a certain period of time. It is presumed that all the tasks have equal priority here. Scheduling based on priorities is further categorized as scheduling of dynamic and static priorities. The priority is allocated to each task during compilation time in static priority scheduling, whereas the priority of each task is resolved during run time in dynamic priority scheduling. Rate Monotonic (RM) and Deadline Monotonic (DM) scheduling are instances of static priority scheduling and Minimal Deadline First (MDF) and Least Laxity Time (LLT) are the dynamic priority scheduling scheme [15].

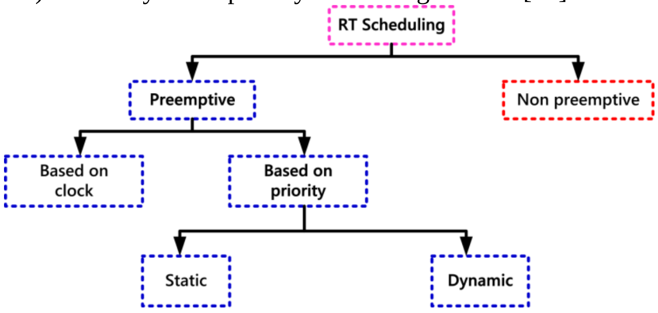


Figure 1 Scheduling Classifications

2.2 Task Model

Consider the set of n periodic, preemptable independent tasks Ʈ = {Ʈ1, Ʈ2, Ʈ3 …Ʈn} such that Ʈk = {Ck, Rk, Dk, Tk} where k =1, 2, …. n and Ck, Rk, Dk, Tk corresponds to computation, release time, deadline and task period. Task utilization is Uk= Ck / Tk and total system utilization is as equation 1.

$$U = \sum_{k=1}^{n} U_k$$

2.3 Minimal Deadline First (MDF)

It is a sort of ideal dynamic schedule in which tasks are prioritized in accordance with their deadline. During the execution of tasks the priority will vary dynamically. With periodic tasks scheduled that have deadlines equivalent to their periods, MDF has a 100 percent usage limit. The schedulability test for MDF is as equation 2.

$$U = \sum_{k=1}^{n} U_k \leq 1$$

Consider the task set as shown in Table 1 consisting of 10 periodic tasks. While performing the scheduling analysis for the set of specified tasks under MDF Scheduling for 4 processors, as shown in Figure 2 we have the scheduling approach.

2.4 Dynamic Queue Minimal Deadline First (DQMDF)

The main objective is to optimize the scheduling process in uniprocessor in order to improve the performance of real-time systems while at the same time providing adequate progress and response to all applications.

The dynamic priority based approaches provide anopportunity to develop an application based on a dynamic approach with some predictability approaches for feasibility. Priority is allocated based on their minimum deadline in the priority-driven preemptive strategy. In this case, before meeting its deadline, the task may be preempted during the execution time. Therefore, in this situation, the timing constraints should be evaluated continuously until the deadline arrives [16].

Table 1. Example Taskset

Task	T_k	D_k	C_k
T1	1.074	1.074	0.15
T2	18.29	18.29	4.06
T3	123.47	123.47	22.18
T4	280	280	23.24
T5	1.68	1.68	0
T6	183.27	183.27	5.53
T7	241.11	241.11	0.73
T8	150.21	150.21	21.79
T9	643.34	643.34	37.24
T10	22.70	22.70	2.27

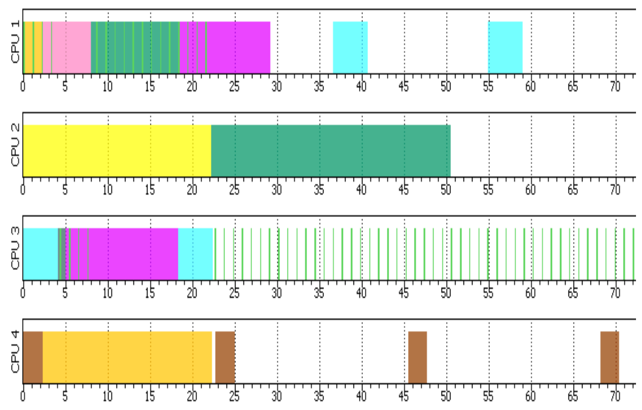


Figure.2 MDF Scheduling in 4 core processor

Initially, each process with a specific amount of tasks is considered for scheduling in the scheduling process. While each process is scheduled at a certain time for execution, the currently available process ready for running is brought in the ready queue. Remaining execution-ready processes are continuously inspected for a threshold value beyond which the task would exceed the execution-time limit. The threshold value for the entire task ready for execution as per equation 2 is continually tracked. The permissible threshold value is 75%. If any of the ready-to-execute processes have a threshold below 75%, the process with the lowest threshold value is preempted with the process available in the ready queue. And according to the threshold values, the priorities are changed so that no task would cross the execution deadline.

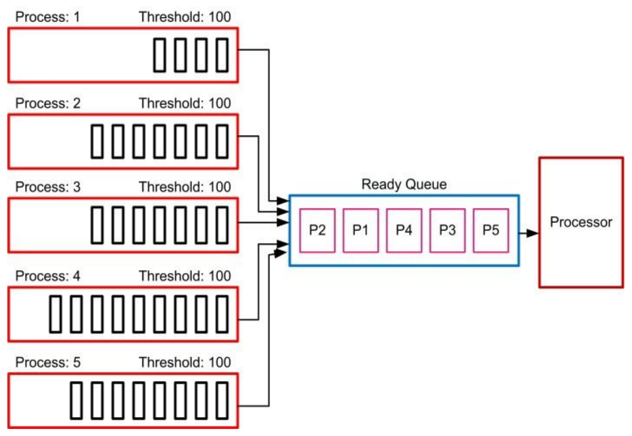


Figure.3 – DQMDF Scheduling Concept – Before preemption

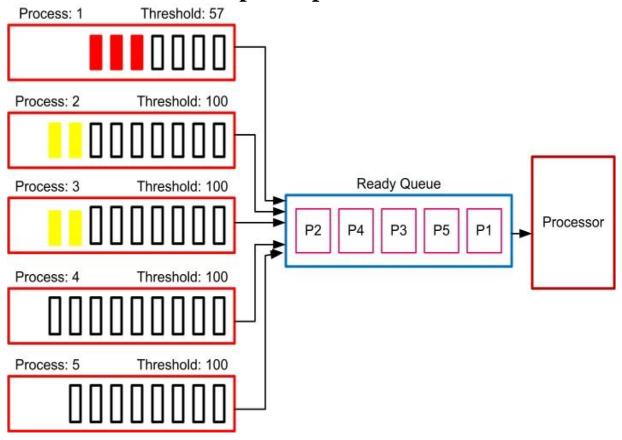


Figure.4– DQMDF Scheduling Concept – After Preemption

Initially, it is suspected that with the particular amount of tasks as shown in Figure 3, each process to be performed in the processor In the run queue, the priority of execution in the processor is scheduled according to the deadline of each of the processes P1, P2, P3, P4, P5 and tasks in the processes. According to these priorities, the process would be executed. Here, the higher priority of execution in the processor is regarded to be P5.

Figure 4 refers to dynamically arriving tasks in each of these processes in the red and yellow color boxes. Without missing the deadline, the yellow color boxes refer to the arriving tasks can be performed at the scheduled time in the process. These processes would have a threshold of 100 percent (which is the proportion of executable tasks without a missing time-limit in specific process) i.e. all tasks would be executed within a scheduled time-limit. In contrast, when the red-colored tasks arrive in processes, the task execution with scheduled priority in the run queue will miss the deadline. Therefore the threshold limit is set as 75%, all processes with dynamic tasks below 75% would be preempted in the processor run queue for execution. From Figure 4, we can see that if executed according to the initial scheduling priority, the process P1 with 3 dynamic tasks arriving would miss the deadline. Thus, after calculating the threshold (which is 57 percent according to the algorithm in section III) for process P1 below the threshold limit, the process in run queue will be preempted and the new priority assigned to the process will be assigned. Process P1 would be given the greatest priority in the processor while all other processes in the run queue would be decreased by one priority value as shown in Figure 4. Therefore, in the run queue, all processes would be executed by priority.

$$T_{Deadline} = T_{Arrival} + (n * T_{exe}) \quad (3)$$

$$Threshold = (100 – (N_{DLMiss} / N_{Task}) * 100 \quad (4)$$

$$T = T_{Deadline} - T_{Arrival} \quad (5)$$

Where TDeadline is the deadline, TArrival is arrival time of a task, Texe is execution time, NDLMiss is the number of deadline miss, NTask is the total number of tasks and T is period of tasks. Assume relative deadline.

The feasibility of schedule is

$$\sum \frac{T_{exe}}{T} < 1 \quad (6)$$

III. ALGORITHM

Algorithm: DQMDF Scheduling		
Step 1	:	Calculate no of processes and Task associated with it to be executed in the cluster using MDF algorithm.
Step 2	:	Calculate deadline as per equation (3).
Step 3	:	Arrival time for each process is stamped for execution within the deadline.
Step 4	:	Calculate threshold as per equation (4)
Step 5	:	If (threshold < 75) Reschedule that process immediately
Step 6	:	Monitor the Threshold value of process continuously.
Step 7	:	Calculate of Period of Task as per equation (5).
Step 8	:	Calculate the feasibility of process as per equation (6)

This is the condition to be followed, if they have to meet the deadlines, not to overload the processor. Figure 5 shows the number of deadline misses for standard MDF versus Dynamic Queue Minimal Deadline First (DQMDF). The comparison of deadline misses for the considered algorithms was virtually simulated in SimSo Simulator using python code. The scheduling algorithm has been implemented to virtually generated tasks in the each of the process with the number of cores for the processor being fixed. The outcomes are presented graphically as shown in Figure 5 by varying the number of tasks for these processes and by scheduling them accordingly. The DQMDF has fewer deadline misses compared to standard MDF, according to the simulation results. This algorithm has therefore demonstrated to be better for real-time operating system execution.

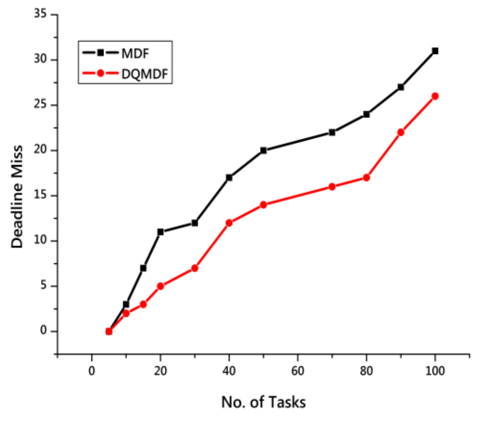
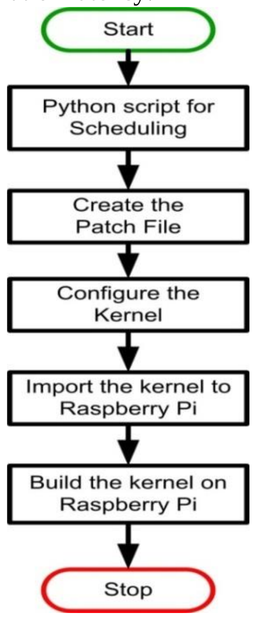


Figure.5 Comparison of MDF and DQMDF

IV. METHODOLOGY

The scheduling algorithm program is initially written as in [17] [18]. This program is developed in the kernel of Raspberry Pi OS (Raspbian) as a module to be updated. It is developed as a patch to be attached as shown in Figure 6 to the Raspberry Pi OS kernel. The OS is then compiled on the host platform with the updated kernel. The kernel would be configured and constructed on Raspberry Pi after effective compilation. Using the cyclic test tool [10], the build kernel is examined for operation latency.



Start
Python script for Scheduling
Create the Patch File
Configure the Kernel
Import the kernel to Raspberry Pi
Build the kernel on Raspberry Pi
Stop

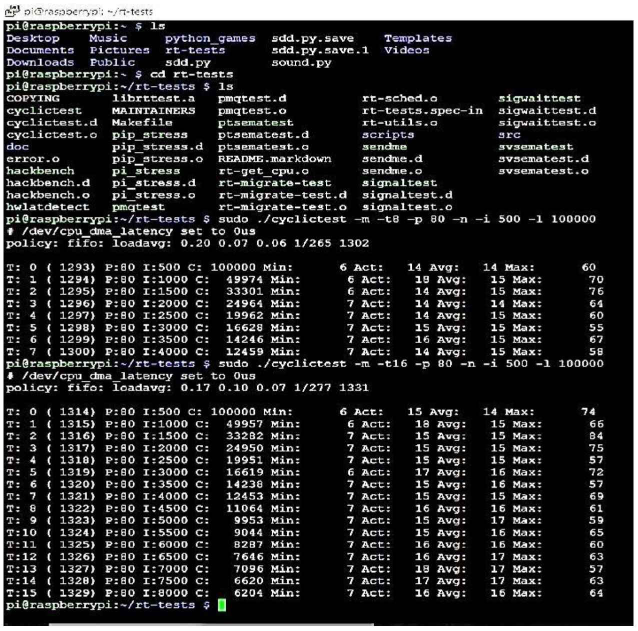
Figure.6 Flowchart for writing kernel code in python

V. HARDWARE SETUP & RESULTS

As shown in figure 7, Raspberry Pi has 4 cores each with one thread. Here 4 processes with variable number of execution tasks are considered. The cyclic testing tool results in a minimum average and peak latency time for pseudo code execution. In the cyclic test tool, the number of loops for performing a task in each process and the interval between the performances are defined. Figure 8 shows the difference in the latency of execution of processes for the normal kernel on Raspberry Pi and the modified kernel with dynamic queue minimal first scheduling algorithm.

Figure 7 - Connection of Multicore system to PC

5.1 Result



```
pi@raspberrypi: ~/rt-tests
pi@raspberrypi:~ $ ls
Desktop    Music     python_games  sdd.py.save    Templates
Documents  Pictures  rt-tests      sdd.py.save.1  Videos
Downloads  Public    sdd.py        sound.py
pi@raspberrypi:~ $ cd rt-tests
pi@raspberrypi:~/rt-tests $ ls
COPYING        librttest.a   pmqtest.d          rt-sched.o          sigwaittest
cyclictest     MAINTAINERS   pmqtest.o          rt-tests.spec-in    sigwaittest.d
cyclictest.d   Makefile      ptsematest         rt-utils.o          sigwaittest.o
cyclictest.o   pip_stress    ptsematest.d       scripts             src
doc            pip_stress.d  ptsematest.o       sendme              svsematest
error.o        pip_stress.o  README.markdown    sendme.d            svsematest.d
hackbench      pi_stress     rt-get_cpu.o       sendme.o            svsematest.o
hackbench.d    pi_stress.d   rt-migrate-test    signaltest
hackbench.o    pi_stress.o   rt-migrate-test.d  signaltest.d
hwlatdetect    pmqtest       rt-migrate-test.o  signaltest.o
pi@raspberrypi:~/rt-tests $ sudo ./cyclictest -m -t8 -p 80 -n -i 500 -l 100000
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.20 0.07 0.06 1/265 1302

T: 0 ( 1293) P:80 I:500 C: 100000 Min:      6 Act:   14 Avg:   14 Max:      60
T: 1 ( 1294) P:80 I:1000 C:  49974 Min:      6 Act:   18 Avg:   15 Max:      70
T: 2 ( 1295) P:80 I:1500 C:  33301 Min:      6 Act:   14 Avg:   15 Max:      76
T: 3 ( 1296) P:80 I:2000 C:  24964 Min:      7 Act:   14 Avg:   14 Max:      64
T: 4 ( 1297) P:80 I:2500 C:  19962 Min:      7 Act:   14 Avg:   15 Max:      60
T: 5 ( 1298) P:80 I:3000 C:  16628 Min:      7 Act:   15 Avg:   15 Max:      55
T: 6 ( 1299) P:80 I:3500 C:  14246 Min:      7 Act:   16 Avg:   15 Max:      67
T: 7 ( 1300) P:80 I:4000 C:  12459 Min:      7 Act:   14 Avg:   15 Max:      58
pi@raspberrypi:~/rt-tests $ sudo ./cyclictest -m -t16 -p 80 -n -i 500 -l 100000
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.17 0.10 0.07 1/277 1331

T: 0 ( 1314) P:80 I:500 C: 100000 Min:      6 Act:   15 Avg:   14 Max:      74
T: 1 ( 1315) P:80 I:1000 C:  49957 Min:      6 Act:   18 Avg:   15 Max:      66
T: 2 ( 1316) P:80 I:1500 C:  33282 Min:      7 Act:   15 Avg:   15 Max:      84
T: 3 ( 1317) P:80 I:2000 C:  24950 Min:      7 Act:   15 Avg:   15 Max:      75
T: 4 ( 1318) P:80 I:2500 C:  19951 Min:      7 Act:   15 Avg:   15 Max:      57
T: 5 ( 1319) P:80 I:3000 C:  16619 Min:      6 Act:   17 Avg:   16 Max:      72
T: 6 ( 1320) P:80 I:3500 C:  14238 Min:      7 Act:   15 Avg:   16 Max:      57
T: 7 ( 1321) P:80 I:4000 C:  12453 Min:      7 Act:   15 Avg:   15 Max:      69
T: 8 ( 1322) P:80 I:4500 C:  11064 Min:      7 Act:   16 Avg:   16 Max:      61
T: 9 ( 1323) P:80 I:5000 C:   9953 Min:      7 Act:   15 Avg:   17 Max:      59
T:10 ( 1324) P:80 I:5500 C:   9044 Min:      7 Act:   15 Avg:   16 Max:      65
T:11 ( 1325) P:80 I:6000 C:   8287 Min:      7 Act:   16 Avg:   16 Max:      60
T:12 ( 1326) P:80 I:6500 C:   7646 Min:      7 Act:   16 Avg:   17 Max:      63
T:13 ( 1327) P:80 I:7000 C:   7096 Min:      7 Act:   18 Avg:   17 Max:      57
T:14 ( 1328) P:80 I:7500 C:   6620 Min:      7 Act:   17 Avg:   17 Max:      63
T:15 ( 1329) P:80 I:8000 C:   6204 Min:      7 Act:   16 Avg:   16 Max:      64
pi@raspberrypi:~/rt-tests $ 
```

Figure 8 Latency Time of Tasks in process

5.2 Performance analysis

By running the RT-cyclic test tool by defining number of tasks to be executed, priority of the execution in each of the processes (tasks are assigned dynamically to each of the processes) corresponding to the cores of the processor with kernel of non-real time scheduling algorithm (i.e. default scheduling algorithm – Completely Fair Scheduling (CFS)) the maximum latency time for some tasks executed in the processor is 150ns to 200ns while others have a very least latency time below 50ns. Therefore, it can be established that if high-priority tasks to be performed can have a high latency time, while low-priority tasks can have a low latency time to be performed. So the execution of high-priority tasks can be delayed. Ultimately, using real-time systems that need to be extremely responsive becomes unfeasible.

Similarly, the maximum latency time was observed by performing the RT-cyclic latency test for tasks executed with the modified kernel (replacing the CFS scheduling algorithm with the DQMDF scheduling algorithm in the kernel scheduling module). For the maximum latency time for each task is observed, the maximum latency time was around 50ns to 80ns for almost all tasks executed with modified scheduling policy. Therefore it is evident that the response time of execution has been lowered while assigning the greater priority to the tasks. So the tasks with high priority can respond earlier for execution and also the tasks with the low priority of execution are responded are with optimal response time without the long delay in execution as compared with non-real time Kernel (with CFS scheduling algorithm). Thus, the DQMDF scheduling algorithm proves to be more responsive in real-time context and can therefore be implemented in the kernel with the non-real-time operating system that can be implemented with real-time systems.

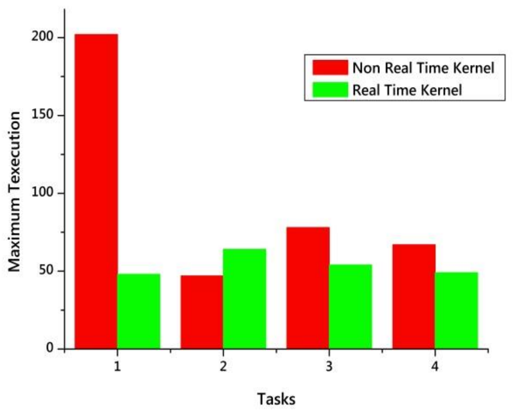


Figure.9 Performance Analysis for 4 tasks

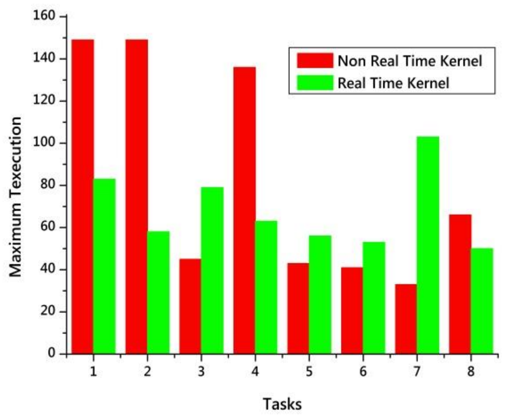


Figure.10 Performance Analysis for 8 tasks

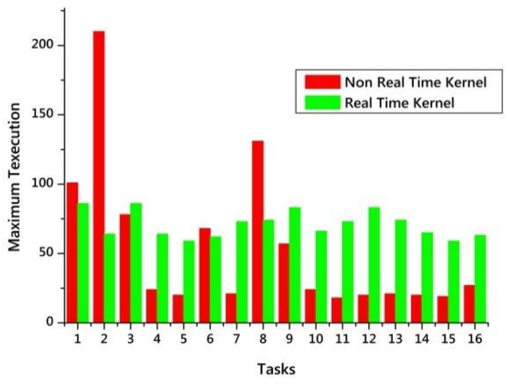


Figure.11 Performance Analysis for 16 tasks

VI. CONCLUSION

The proposed DQMDF is implemented in Simso in this paper by considering 4 cores and comparing the performance of the proposed one to the standard MDF. Compared to state of the art work, the deadline miss is proven to be less in the suggested technique. Then the same at the hardware level in the non-real time kernel is implemented in 4 core processor, it has been shown that the latency time is reduced considerably. We plan to implement that enormous task set in the future.

REFERENCES

1. Senthilkumar, M. "Energy-Aware Task Scheduling Using Hybrid Firefly-BAT (FFABAT) in Big Data." Cybernetics and Information Technologies, vol. 18, no. 2, 2018, pp. 98–111., doi:10.2478/cait-2018-0031.
2. Kandasamy, Nagarajan, et al. "Task Scheduling Algorithms for Fault Tolerance in Real-Time Embedded Systems." Dependable Network Computing, 2000, pp. 395–412., doi:10.1007/978-1-4615-4549-1_18.
3. Ha SW, Lee YK, Vu TH, Jung YJ, Ryu KH. An environmental monitoring system for managing spatiotemporal sensor data over sensor networks. Sensors (Basel). 2012;12(4):3997–4015. doi:10.3390/s120403997.
4. Cain C, Haque S. Organizational Workflow and Its Impact on Work Quality. In: Hughes RG, editor. Patient Safety and Quality: An Evidence-Based Handbook for Nurses. Rockville (MD): Agency for Healthcare Research and Quality (US); 2008 Apr. Chapter 31.
5. Gao, Huanli, and Yonggui Liu. "Event-Driven Real-Time Actuator Scheduling Strategy over Wireless Sensor and Actuator Networks." Proceedings of the 33rd Chinese Control Conference, 2014, doi:10.1109/chicc.2014.6896663.
6. Sullins, John, "Information Technology and Moral Values", Metaphysics Research Lab, Stanford University, 2019.
7. Di Liu, Nan Guan, Jelena Spasic, Gang Chen, Songran Liu and Todor Stefanov, "Scheduling Analysis of Imprecise Mixed-Criticality Real-Time Tasks," in IEEE Transactions on Computers, vol. 67, no. 7, pp. 975-991, 1 July 2018.
8. N. Romanova, N. Crosby, and V. Pilipenko, "Relationship of Worldwide Rocket Launch Crashes with Geophysical Parameters," International Journal of Geophysics, vol. 2013, Article ID 297310, 15 pages, 2013. https://doi.org/10.1155/2013/297310.
9. Malik, Sehrish, Shabir Ahmad, Israr Ullah , Dong Hwan Park and DoHyeun Kim "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, no. 8, 2019, p. 1- 21.
10. Saifullah, Abusayeed, et al. "Multi-Core Real-Time Scheduling for Generalized Parallel Task Models." 2011 IEEE 32nd Real-Time Systems Symposium, 2011, doi:10.1109/rtss.2011.27.
11. RAS, PH D. JIM. Scheduling Algorithms for Real-Time Systems. LULU COM, 2016.
12. Kohutka, Lukas, and Viera Stopjakova. "A Novel Hardware-Accelerated Real-Time Task Scheduler Based on Robust Earliest Deadline Algorithm." 2018 13th International Conference on Design &amp; Technology of Integrated Systems In Nanoscale Era (DTIS), 2018.
13. Yim, Yin-Goo, and Hyun-Wook Jin. "Analysis of Impact of Multi-Core CPU Bandwidth in Docker Containers." KIISE Transactions on Computing Practices, vol. 24, No. 12, 2018, pp. 675–680., doi:10.5626/ktcp.2018.24.12.675.
14. Arshad Iqbal, Asia Zafar and Bushra Siddique , "Dynamic Queue Deadline First Scheduling Algorithm for Soft Real Time Systems." Proceedings of the IEEE Symposium on Emerging Technologies, 2005., doi:10.1109/icet.2005.1558906.
15. Khera, Ishan, and Ajay Kakkar. "Comparative Study of Scheduling Algorithms for Real Time Environment." International Journal of Computer Applications, vol. 44, No. 2, 2012, pp. 5–8., doi:10.5120/6233-7797.
16. Ruan, Youlin, et al. "Energy-Aware Scheduling for Multicore Real-Time Systems." Management Innovation and Information Technology, 2014, doi:10.2495/miit130391.
17. Abeer Hamdy, Ahmed E. Youssef and Reda Ammar, "Real-Time Workload Allocation on a Uni-Processor." International Journal of Computer Applications, vol. 53, No. 12, 2012, pp. 17–24.
18. M. Kaladevi and Dr. S. Sathiyabama "Comparative Study of Scheduling Algorithms for Real Time Task", International Journal of Advances in Science and Technology, Vol. 1, No. 4, 2010.