International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 /doi:10.14445/22315381/IJETT-V69I9P208

Volume 69 Issue 9, 56-65, September, 2021

Original Article

Adaptive Scheduling Technique Based Operating System for Wireless Sensor Networks and Internet of Things

Anita Patil[1,*] and Rajashree.V.Biradar[2]

[1]*Department of CSE, Ballari Institute of Technology and Management, Bellary-583104, Karnataka, India*
[2]*Department of CSE, Ballari Institute of Technology and Management, Bellary-583104, Karnataka, India*

[1]anitha.bijapur@gmail.com, [2]rajashreebiradar@yahoo.com

Abstract *- Wireless Sensor Network (WSN) has significance in various fields, including home and industry automation, medical instrumentation, military surveillance, etc. Though the battery-dependent and resource-constrained tiny sensor nodes challenge the design of the operating system (OS) very critically, there are many OS that exists for WSN and IoT. However, the available operating systems have their own advantages and disadvantages for various WSN applications. Among those, TinyOS is the widely used, highly documented, and most suitable OS for low power devices. Conversely, having only First Come First Serve (FCFS) scheduler is the major limitation of TinyOS that hinders the application developers from using this. The necessity of other schedulers is justified in the motivation part of the paper. Thus, to overcome this problem, the new adaptive scheduling algorithm proposed in this paper provides a choice for FCFS, Priority, and Round Robin schedulers. Moreover, the priority scheduler itself can represent the schedulers SJF, EDF, and any application requirement-based prioritizing scheduler. Accordingly, the application developer can adapt any scheduler for the application. This changing order of tasks' execution also benefits the overall system performance by giving reduced average waiting for time (AWT) and average turnaround time (ATT), resulting from inefficient utilization of resources and better throughput.*

Keywords *- IOT operating systems, TinyOS Scheduling techniques, Wireless Sensor Network, WSN Applications, WSN operating systems.*

I. INTRODUCTION

Wireless Sensor Network (WSN), being a very special type of network, has many applications in different fields of technology and also is the basis for advanced technologies like IoT. The main specialties of this network are communicating wirelessly and sensing the surrounding environment with the help of tiny sensor nodes. These 2 features are tremendously advantageous as WSN applications also cover the fields where the physical presence of the human being is either impossible or not feasible. Here is an example of industrial automation where in some sort of industry, the working environment temperature may be too high, and there may be life-threatening hazardous operating processes, etc., because of which the physical presence of human beings at that place is not feasible. Another example is medical instrumentation, where the tiny instruments of medical diagnosis may need to pass through human body organs. Like this, there are numerous applications where sensor nodes are working on behalf of the human being. Moreover, wired communication is not at all practical in the densely deployed numerous sensor nodes in the application area [1]. But this wireless communication and the tiny-sized nature of the sensor nodes themselves challenge the design of the Operating System (OS) very critically. Despite that, there are many OS exist for WSN as of today. Some of them are application-specific, some are hardware-specific, etc. For e.g., RTOS (real-time operating system) is meant for real-time applications, raspbian is for the only raspberry pi platform, RIOT (Real-time OS for IoT) is a real-time OS for IoT, and so on. Thus, none of them is general purpose. Such many issues make the operating system of WSN itself a research issue.

Basically, there are many research issues in WSN, including operating systems of WSN, uninterrupted support for diversified numerous applications of WSN [2, 3], energy efficiency, routing in WSN, and so on. The novel work presented in this paper is for the OSs of WSN and IoT. By the way, WSN being the backbone support of IoT technology, share the same set of OSs with IoT [4, 5, 9]. The comparative analysis of some popular OSs: TinyOS, Contiki, RIOT, freeRTOS, MANTIS, and SOS concludes that TinyOS is the open-source, most robust, innovative, traditional, highly documented, and widely used OS. Moreover, this is the most suitable OS for low power devices, which is the main concern for energy efficiency [6, 7, 8, 13]. Being energy efficient means a lot as the densely deployed and resource-constrained tiny nodes have to survive for longer in application fields to achieve their

purpose [19]. Because the unattended and battery-dependent life span of the nodes decides the effectiveness of the application. For example, the battery life of the mica2 mote while running the blink application in different OS, namely TinyOS, MANTIS OS, and SOS, are respectively 22.49 days, 7.84 days, and 7.73 days (approximately) [10].

Ultimately, TinyOS is the OS for resource-constrained, low-power tiny devices being used in various applications of both WSN and IoT [11, 12, 13]. Though with this major concern towards energy efficiency [12] and less memory footprint (less than 400 bytes), TinyOS has the disadvantage in scheduling, as it has only a First Come First Serve (FCFS) scheduler [14, 15, 16, 17]. In FCFS, the tasks get processed based on their arrival order that can be appropriate for some kind of applications or at some situation only. Although this is one of the best scheduling algorithms, it can't fulfill the requirements of all types of tasks and applications. Thus, having only the FCFS scheduling technique affects other parameters also as not supporting real-time applications, reduced performance of some tasks, inefficient usage of resources, etc. Instead, if the operating system is flexible in scheduling by having multiple schedulers like priority scheduler, round-robin scheduler, the shortest job first scheduler, etc., then it is more beneficial for the tasks, applications, as well as resource utilization. In this direction, we have surveyed [18] the recent literature for other possible scheduling algorithms for TinyOS. Then, came to the conclusion of designing and integrating the new adaptive scheduling algorithm that allows the application developer to adapt a suitable scheduler from the list of FCFS scheduler, Priority scheduler, and Round Robin scheduler.

In this regard, the first section of this paper introduces WSN applications' requirements and TinyOS along with FCFS scheduling, and then the second section articulates the compulsion of other types of schedulers, thereby conveying the scope of this novel work. The new adaptive scheduling algorithm implementation is explored in the 3rd section, followed by the results and discussion in the 4th section.

II. MOTIVATION

In this technological era, where every field of life is evolved by technologies like WSN, IoT, robotics, artificial intelligence, and machine learning, etc. Consequently, the daily life needs are getting fulfilled through one or the other hardware appliance, which in turn run by software technology. An operating system is a basis for such appliances along with application-specific software. Specifically, OS of WSN and IoT are very challenging as they have to reside in limited memory, then control and coordinate the constrained resources of the tiny sensor node [5, 19]. Moreover, WSN and IoT cover a wide range of distinct fields with the applications like seismic detection, military surveillance, wildlife study, underwater study, medical instrumentation, industrial automation, etc. The varying requirements of such diversified applications resulted in the number of OSs with specific features, like real-time application supporting OS, platform-specific OS, energy-efficient OS, etc. This causes the application developer to study all these OS in detail while selecting a suitable one for new application development. This makes the application developer invest the time and put more effort, along with the intended application designing. Thus, here is the necessity of surveying the existing OS, then improvise the best OS among them as a generalized OS that can be a default selection to cover a wider range of applications.

Based on the requirements of diversified WSN applications [2, 3], the sensor node can have multiple tasks like light sensing, sound sensing, vibration sensing, temperature sensing, data processing, communicating to the base station, etc. For example, in wildlife study, if the application is for capturing the images of wild animals, monitoring the surrounding environment, and recording the video of any event occurrence like wandering of an animal, fighting of animals, or the activities of an animal when it is alone, then different sensors like light sensing, sound sensing, temperature sensing, etc., need to function like the tasks. At the same time, other tasks include processing the gathered data from different sensors, aggregating the data, and communicating to the base station. Thus, when there are multiple tasks involved in an application, then definitely the order of their processing will have an impact on the application result.

As already said, the event-driven TinyOS has one and only FCFS scheduler that processes the tasks in the order of their occurrences as a natural practice, but the parallel and continuous tasks may need a change in their processing order in some situations. For instance, in wildlife study applications, if the temperature exceeds the defined threshold limit, then temperature sensing, processing this sensed data, and communicating this processed data to the base station must be given the highest priorities compared to all other tasks. Thereby predicting the forest fire, the fire accident can be prevented. For such an application priority scheduler is beneficial. In some applications where scheduling is done using SJF scheduler, there the average waiting time (AWT) and the average turnaround time (ATT) of all tasks will be lesser satisfying all tasks as well as the application. Moreover, such scheduling improves the overall system performance by utilizing all the resources efficiently [19]. While in some applications giving equal opportunity to each task in each round may be the requirement. Such type of scheduling is the responsibility of the round-robin scheduler (RR). At the same time, the RR scheduler gives the least possible response time to each one of the tasks giving an illusion of interactive task processing. Actually, this is also the better option for TinyOS as it is not having real-time application support. Like this, there are different

options for scheduling a single processor among multiple tasks, and those have to get streamlined in this new adaptive scheduling algorithm. This is the main motivation behind this novel work.

On the other hand, authors in [20] say that there are abundant applications of WSN and IoT, for which specialized OSs are needed, but a slow reaction of OS researchers is an alarm for the urgency of more research in this area. However, in reality, OS developers/researchers are rare due to the fact that it is a highly specialized field with a very slow curve and tolerance for change. This is one more motivating point for this research. In this direction, authors in [8] did a detailed survey in 2016. As per the survey, TinyOS alone is in 60% usage, and the remaining all OS together is in 40% usage [8]. The same is depicted below Figure1.

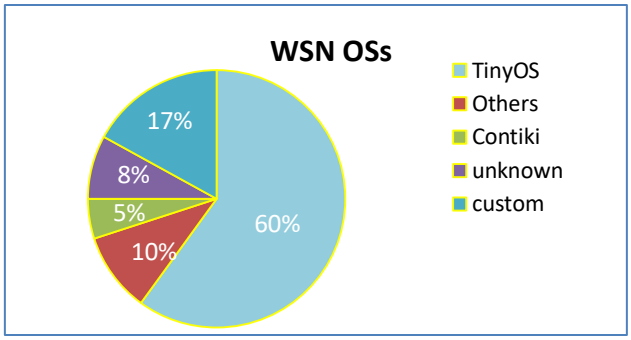

Fig. 1Usage of different OSs (adapted from[8])

In this situation where TinyOS itself is covering more than 50% of the WSN and IoT applications, then here is the necessity of research in TinyOS to nullify its disadvantages.

The detailed study of TinyOSreveals the advantages and disadvantages that motivated us to proceed further in the field of processor scheduling among multiple tasks of the application. Further, the detailed survey on "Scheduling Techniques for TinyOS" [18] in 2016 concludes that there is the possibility of designing and implementing a new algorithm that can retain the existing FCFS scheduler and also provide other important schedulers.

In fact, the TinyOS developers themselves provided the document in the "docs" folder of TinyOS documentation to encourage researchers to design and integrate new schedulers in TinyOS. The document is available as TinyOS Enhancement Proposal-106 (TEP-106).

- 2.55*1.24*0.24 inches, within which 10kb RAM, 48kb flash memory, 2*AA batteries, 8MHz MSP430 microcontroller, 3 sensors, and 3 LEDs, etc., objects are soldered, which all together weighs 23 grams(excluding batteries weight)[23,25].

Thus, there are many motivating factors behind this empirical work of the new adaptive scheduling algorithm that retains the existing FCFS scheduler and also provides 2 more schedulers in choice.

III. IMPLEMENTATION

The main work has the flow as shown in the flowchart of Figure-2. Here, the job queue is nothing but the flash memory of the sensor node. For example, Telosb node has 48kb flash memory and 10kb RAM. For better utilization of the main resources in the node, which is input-output devices and processor (herein TinyOS, processor or CPU is nothing but MSP430 microcontroller), the good combination of I/O (Input/Output) bound tasks and processor bound tasks are to be placed on RAM. If these tasks don't need any fashion of execution, then the pre-existing FCFS scheduler itself can schedule the processor based on their arrival order; else, the appropriate one from the new adaptive scheduling algorithm schedules the processor. This new adaptive scheduling algorithm provides the choice for FCFS, Priority, and Round Robin (RR) schedulers.

The new adaptive scheduling algorithm implemented in this novel work is named AdaptiveSchedulerC.nc. As the name itself indicates, this scheduler allows an application developer to adapt any of the schedulers as per requirement. Adaptive scheduling algorithm provides choice among 1, 2, and 3 for FCFS, Priority, and Round Robin schedulers, respectively. The application developer has to enter the choice taken in the header file named SchedulerSelection.h. If the priority scheduler is the choice taken, then the priorities for tasks also must be entered in the SchedulerSelection. Header file itself.

This newly designed and developed adaptive scheduling algorithm implementation is carried out as follows.

- Here in this work, tinyos-2.1.2 is installed in Ubuntu 18.04. It can be installed in the Windows system also. TinyOS has a footprint of fewer than 400 bytes, which is the core or base code of OS that has to fit in node memory along with the compiled code of the application and other required software.

- Telosb has MSP430 microcontroller. Hence, the emulator used here is MSPSim [21, 22].

- In the emulator, Telosb is the platform [14, 24] used, which is one of the suitable sensor boards for TinyOS. Like any sensor node, the size of Telosb is also tiny i.e
- The language nesC [26] is used to code an Adaptive scheduling algorithm.
- The interfaces Scheduler, TaskBasic, and McuSleep of the "tos" folder are redefined to implement the new Adaptive scheduling algorithm.

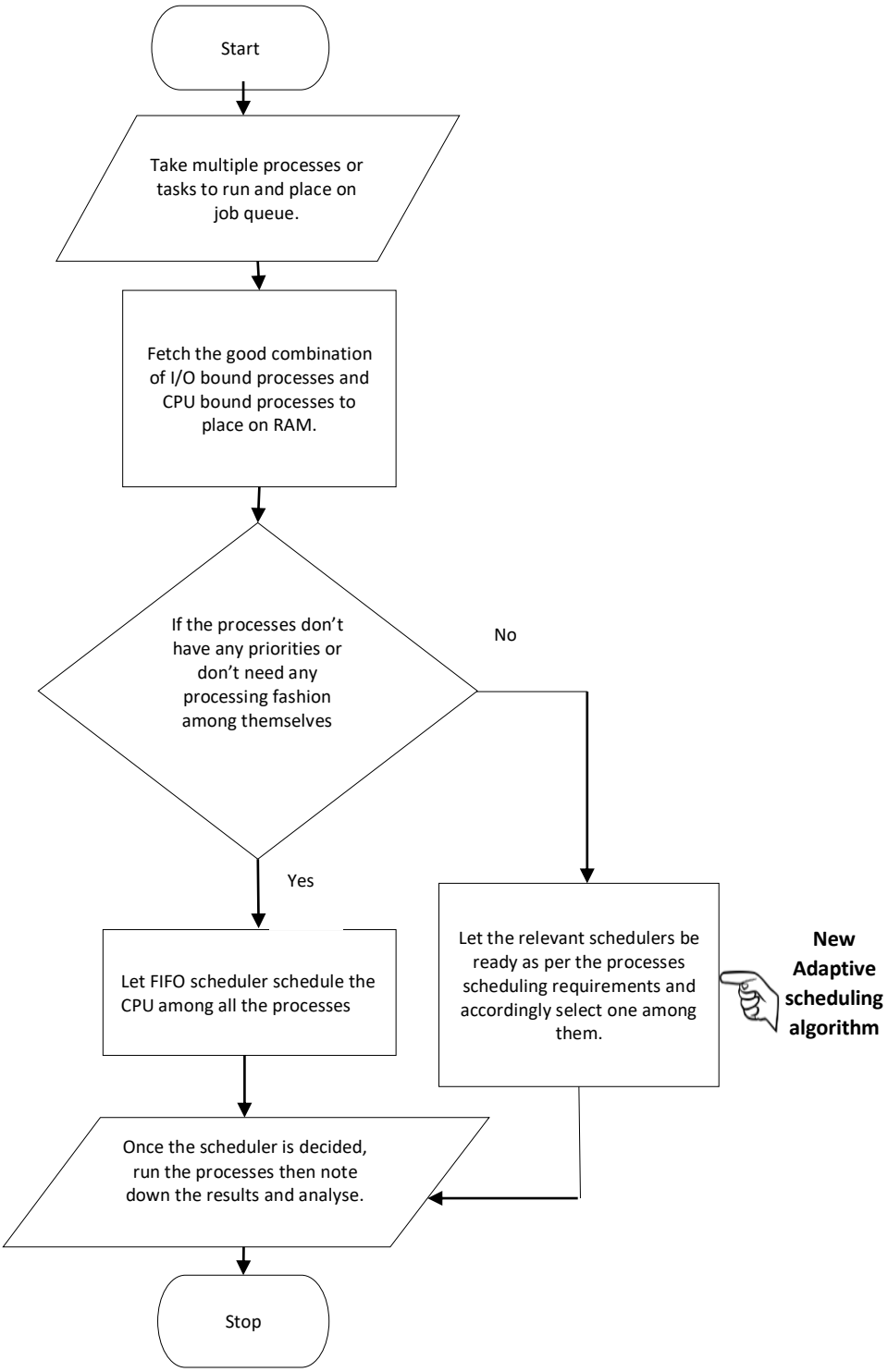

Figure 2. Workflow

The below figure, Fig 3, shows the partial hierarchy of TinyOS-2.1.2 with the newly added schedulers.

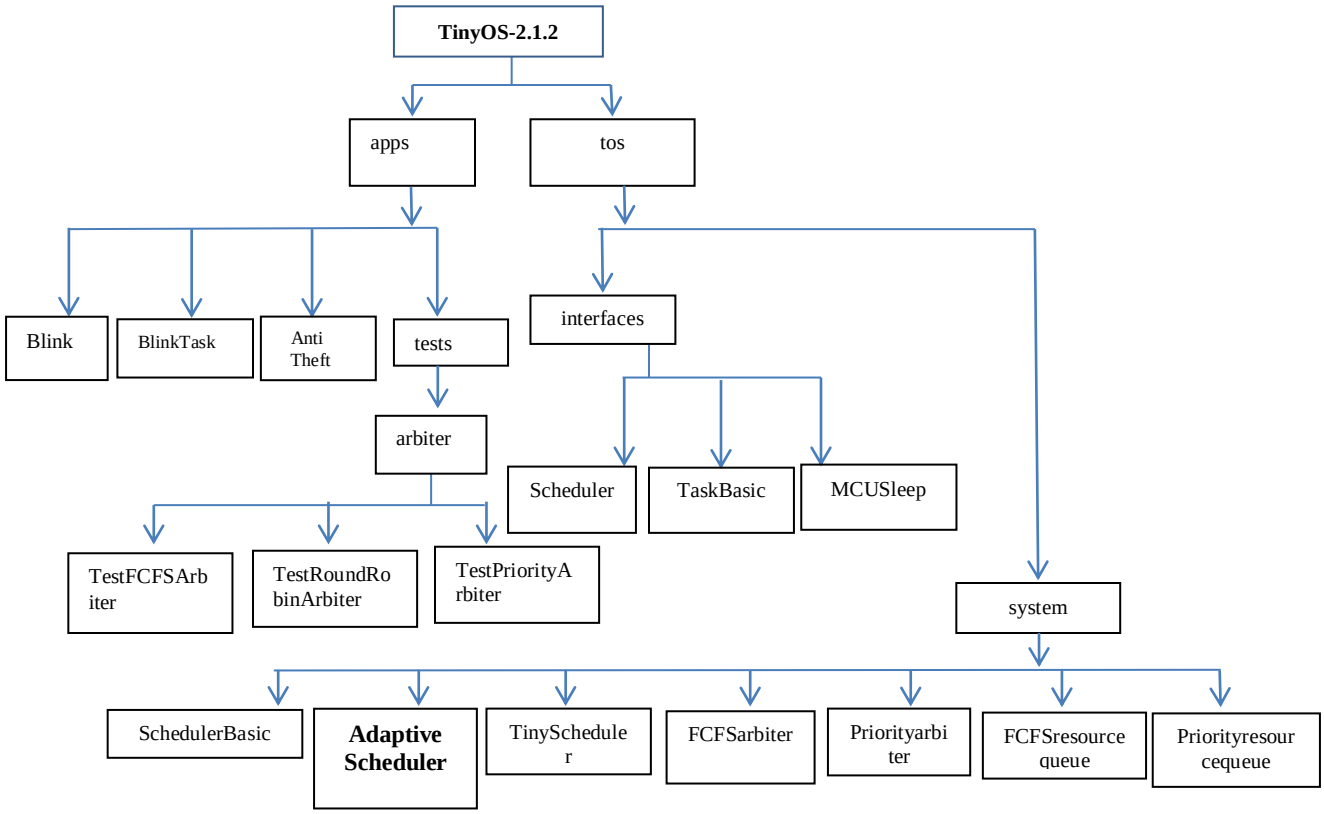

Fig. 3 Partial hierarchy of TinyOS-2.1.2

Basically, in TinyOS-2.1.2, at the different hierarchy, there are many folders like tos (core OS), app (example applications), docs (documentation), system, interfaces, lib, platforms, etc. [27]. In the docs folder, there are around 40 TEPs (TinyOS Enhancement Proposals) [28], TEP-106 says about scheduler in TinyOS. The folder "tos" contains the core of the operating system, which is dispersed in different sub-folders like system, interfaces, tools, platforms, etc. As shown in Fig-3, the new Adaptive scheduling algorithm is implemented in /tos/system/AdaptiveschedulerP.nc, with the help of the interfaces namely Scheduler, TaskBasic, and McuSleep of **tos** folder.

IV. RESULTS AND DISCUSSION

The results of FCFS, Priority, and RR Schedulers are shown respectively in Figure-4 to Figure-6 below.

Each snapshot shows:

• **Application code editor:** - To see the tasks posting order.

• **Control UI:-** The user interface to control the node, like stop and run the execution.

• **Mote GUI:**-GUI with blinking LEDs, sensors, MSP430 microcontroller, etc.

• **Serial mon for MSPsim:**-To sees the order of execution with some printf statements, as there are only 3 LEDs and the number of tasks may exceed 3. Moreover, taking the results from these printf statements is easier than monitoring the blinking LEDs.

All these 3 schedulers are tested for the same application. To read the results of tasks' execution order, the colors-Red1, Green1, Blue1, and Pink1 are displayed inside the tasks test0, test1, test2, and test3, respectively. In task2, the searching function is written to observe the task processing that also executes correctly. As shown in the code editor of the below screenshots, for all three schedulers, the application is the same that posts the tasks with the order test3, test2, test0, test1.

As shown in the below screenshot of Fig-4, the FCFS scheduler schedules the processor to the tasks in the same order of their posting, i.e., First Come First Serve, resulting in the display statements for PINK1, BLUE1, RED1, GREEN1 at MSPSim's serial monitor.

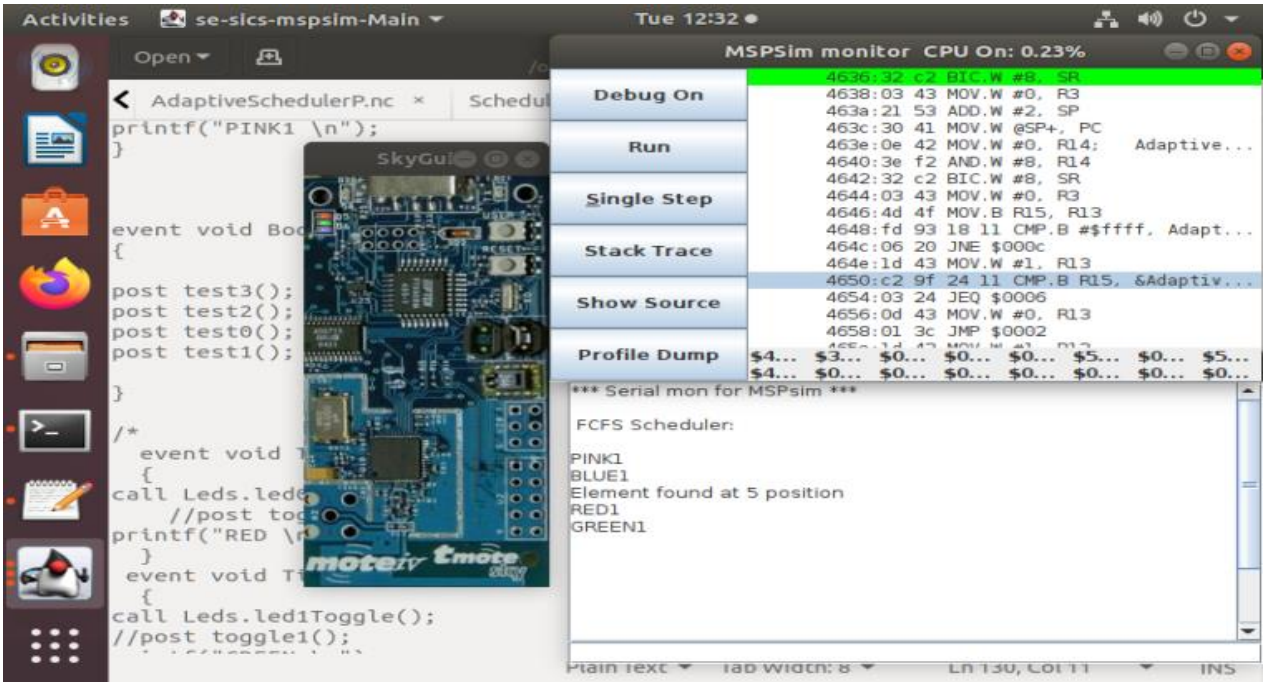

Fig. 4 FCFS scheduler

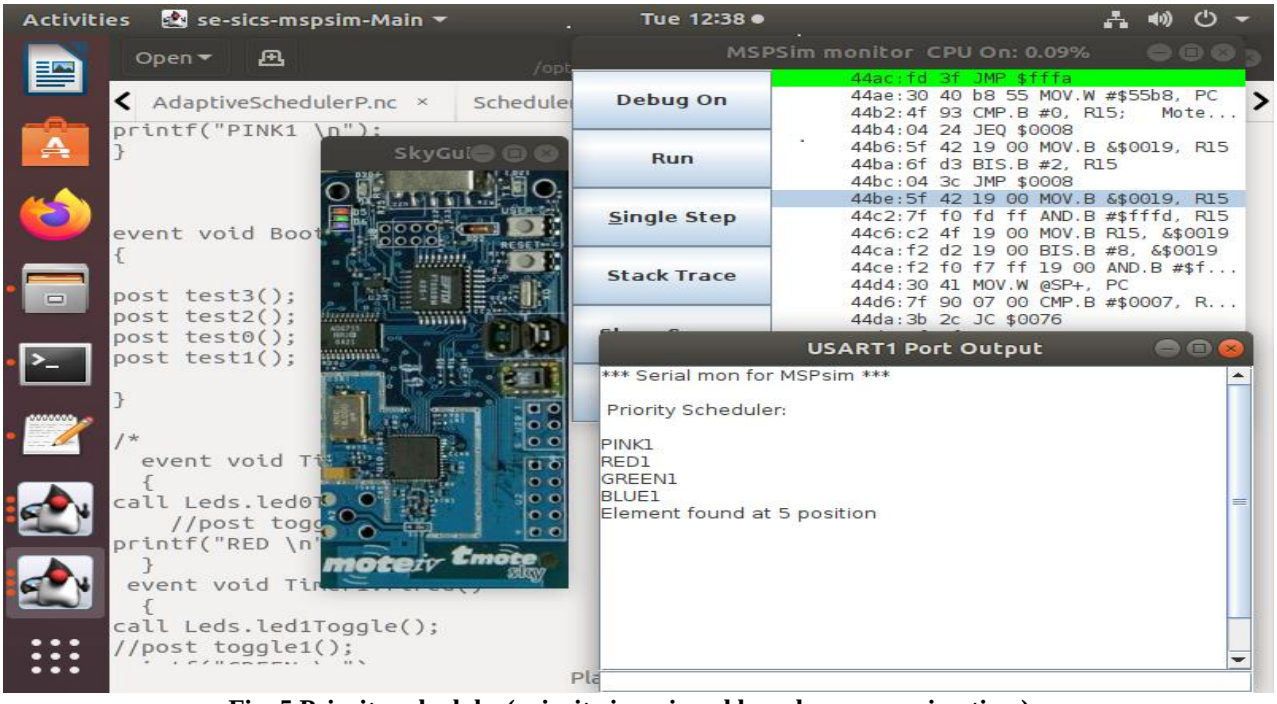

Fig. 5 Priority scheduler(priority is assigned based on processing time)

As shown in the above screenshot of Fig-5, the priority scheduler schedules the processor to the tasks as per priorities assigned to them in SchedulerSelection.h. The largest number indicates the highest priority. With this notion, priorities assigned are 4→test3, 3→test0, 2→test1 and 1→test2 resulting in display statements forPINK1, RED1, GREEN1, BLUE1 at serial monitor of MSPsim.

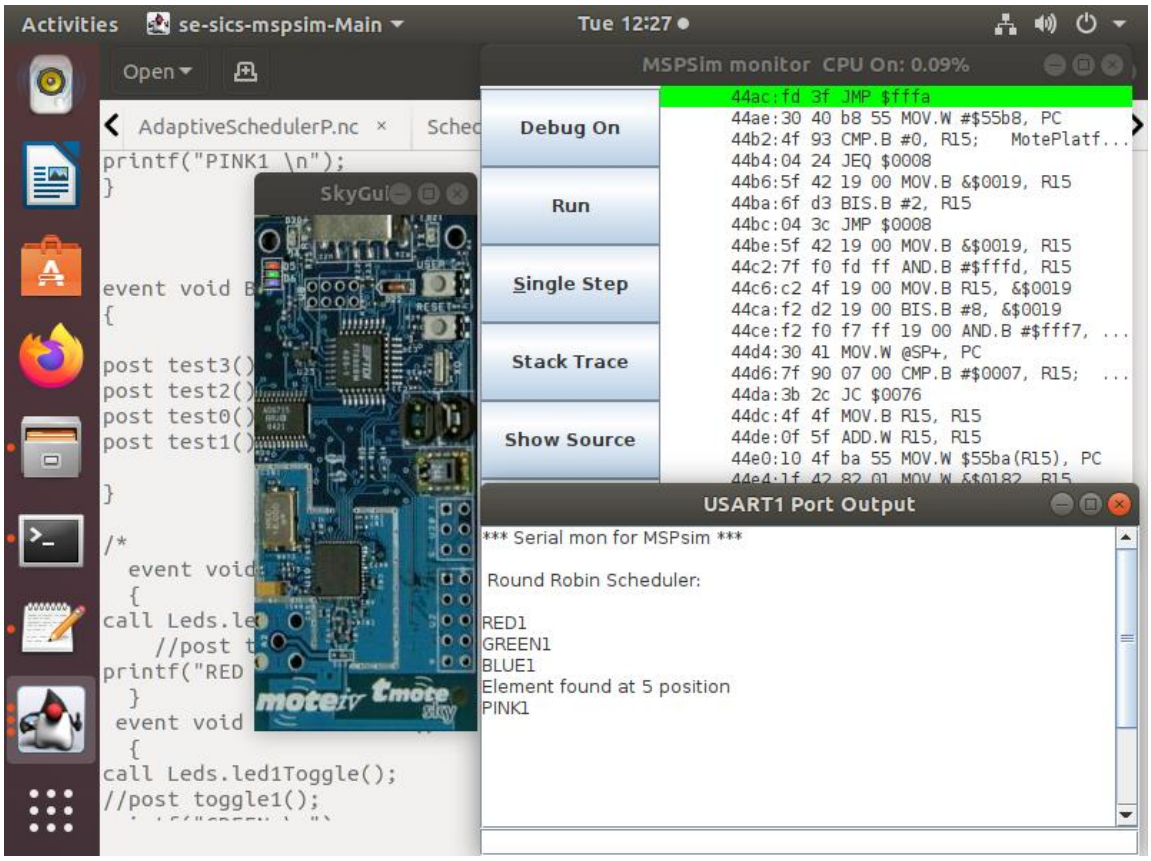

Fig. 6 Round Robin (RR) scheduler

The above snapshot of figure-6 shows RR scheduler results. In RR scheduling, at every round, every task will get a chance to get processed by the processor. It schedules the processor to the tasks in the order of their IDs generated during their definition coding in the application program. In the given program, the tasks are defined in the order test0, test1, test2, test3, and hence, the IDs generated are 0,1,2,3, respectively. Accordingly, the displays are for RED1, GREEN1, BLUE1, PINK1.

Like this, an adaptive scheduling algorithm permits the application developer to adapt the necessary scheduler for the application. This change in task order not only satisfies the tasks but also improves the overall performance of the system with the best utilization of all the resources.

Resource utilization or scheduling algorithm performance can be measured in terms of Average Waiting Time (AWT) and Average Turnaround Time (ATT) of tasks. The lesser the AWT and ATT are, the better the performance [29]. The below-shown tables and graphs of all 3 figures illustrate a theoretical example that runs 4 tasks with the depicted arrival times and processing times. The performance analysis of FCFS scheduling, Priority scheduling, and RR scheduling are respectively explored by figures 7, 8, and 9. Tables display the arrival time, processing time, waiting time, and turnaround time for each one of the tasks along with AWT and ATT. The same is depicted in their respective graphs.

Tasks	Arrival timeless	Processing time msec	Waiting Time msec	Turnaround Time msec
Task 1	0	2	0	2
Task 2	1	4	1	5
Task 3	2	1	4	5
Task 4	3	3	4	7
			AWT= 2.25 msec	ATT= 4.75 msec

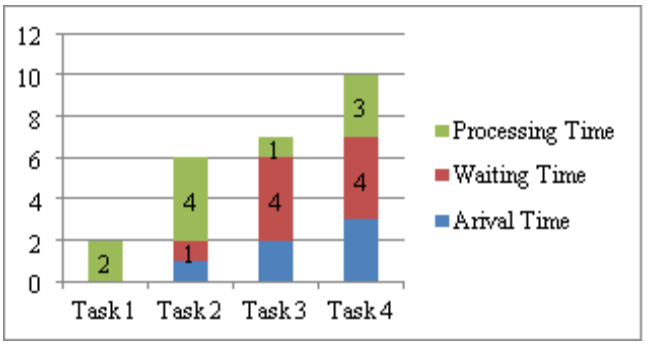

Fig. 7 Performance analyses in FCFS scheduling

Tasks	Arrival time msec	Processing time msec	Waiting Time msec	Turnaround Time msec
Task 1	0	2	0	2
Task 2	1	4	5	9
Task 3	2	1	0	1
Task 4	3	3	0	3
			AWT= 1.25 msec	ATT= 3.75 msec

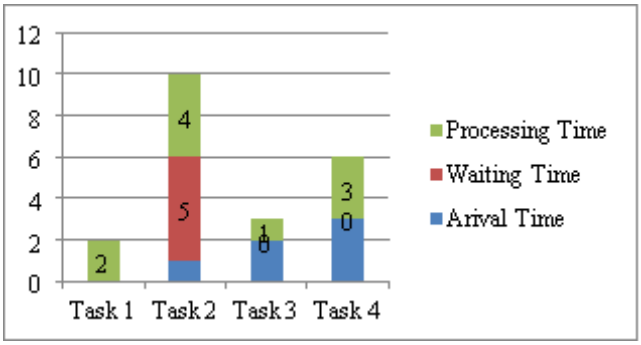

Fig. 8 Performance analysis in Priority scheduling (priority is assigned based on processing time)

Tasks	Arrival time msec	Processing time msec	Waiting Time msec	Turnaround Time msec
Task 1	0	2	3	5
Task 2	1	4	5	9
Task 3	2	1	0	1
Task 4	3	3	3	6
			AWT= 2.75 msec	ATT= 5.25 msec

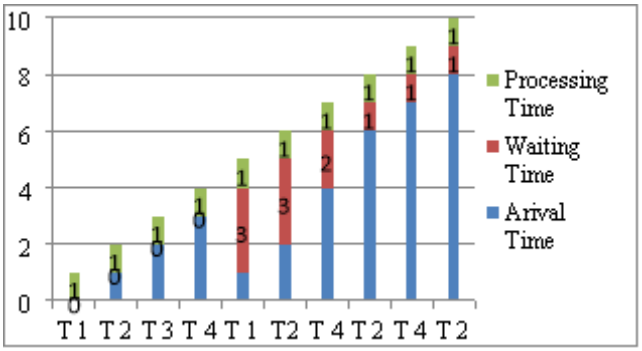

Fig. 9 Performance analyses in RR scheduling

In the RR scheduling graph, the tasks' names are taken as T1, T2, T3, and T4 instead of Task1, Task2, Task3, and Task4, so that they can fit in the chart area. In priority scheduling, the priorities assigned to the tasks Task1, Task2, Task3, and Task4 are respectively 2, 4, 1, and 3. In this example, priority is based on the processing time of the tasks. The task with the least processing time gets the highest priority, and here in this example, the smallest number represents the highest priority. Though, instead of Task3 of priority 1, task Task1 gets executed first.

The reason is, at 0thmsec, only Task1 arrived and start getting processed. Since it is the non-preemptive priority scheduling, Task1 gets completely executed without preempting in between, even when high-priority tasks arrive. By the end of Task1 execution, the remaining all 3 tasks arrive. Then scheduling continues based on their priorities, i.e., Task3, Task4, and Task2. Thus priority assignment can be done based on any criteria such as Early Deadline First (EDF), foreground tasks first, interactive tasks first, etc.

The comparative analysis of the performance of all 3 scheduling algorithms is explored in below Table-1 and the graph of Fig-10**.** The resultant AWT shows that the priority scheduler is the best with nearly half of the AWT of FCFS, whereas the AWT of RR is more than both FCFS and priority scheduling, but important here is that the order of tasks processing is as per the requirement of RR scheduler. Correspondingly ATT also has the same influence, which is, ATT of priority scheduler is least, and that of RR is more than both priority and FCFS schedulers.

Table.1 Performance analysis of FCFS, Priority, and RR scheduling algorithms.

Tasks	Arrival time msec	Processing time msec	AWT FCFS msec	AWT Priority msec	AWT RR msec	ATT FCFS msec	ATT Priority msec	ATT RR msec
Task 1	0	2	0	0	3	2	2	5
Task 2	1	4	1	5	5	5	9	9
Task 3	2	1	4	0	0	5	1	1
Task 4	3	3	4	0	3	7	3	6
			AWT= 2.25 msec	AWT= 1.25 msec	AWT= 2.75 msec	ATT= 4.75 msec	ATT= 3.75 msec	ATT= 5.25 msec

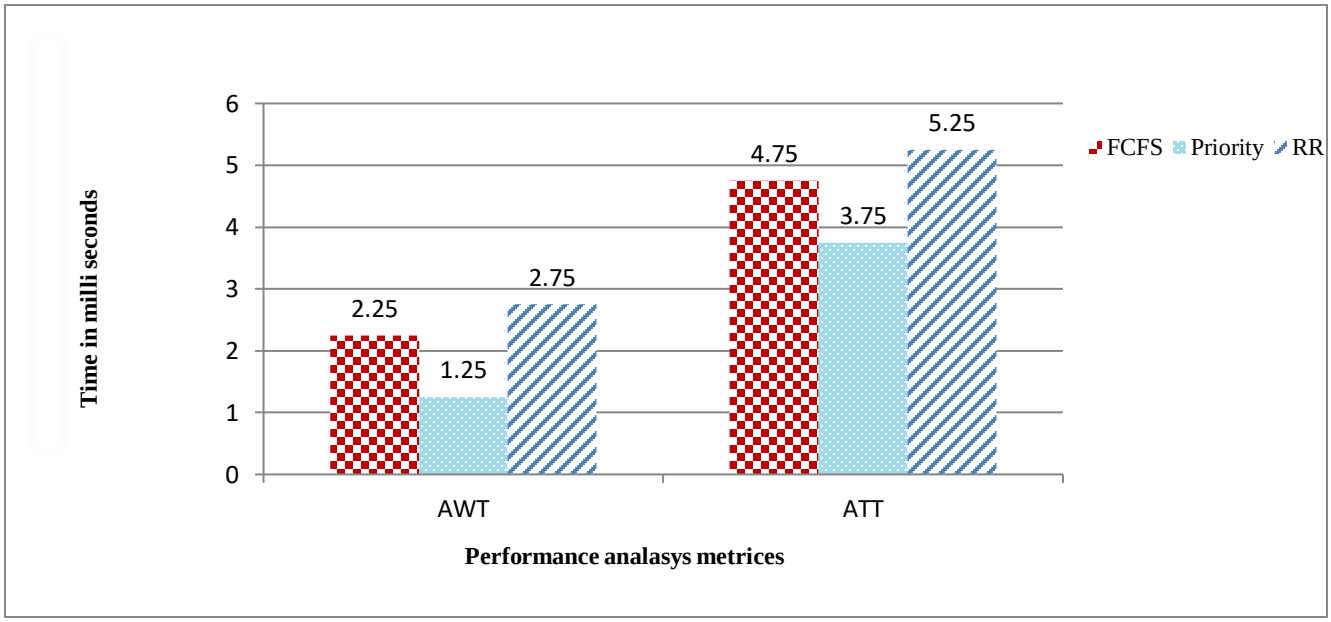

Fig. 10 Performance analysis of FCFS, Priority, and RR scheduling algorithms.

V. CONCLUSION

For decades many researchers working on energy efficiency in WSN declare that tinyOS is the best OS for resource-constrained and low-power tiny devices being used in WSN and IoT applications. Even knowing this fact, application developers hesitate to use this OS as it has only an FCFS scheduler that can't support the diversified applications whose needs may be either SJF or RR or any other scheduling. Nonetheless, now the Adaptive scheduling algorithm proposed in this paper provides choice for FCFS, Priority, and RR schedulers, thereby inspiring the application developer to use TinyOS to get the benefit of energy efficiency nature. At the same time, the priority scheduler itself can represent the schedulers SJF and EDF by assigning priorities to the tasks based on different lengths and different deadlines, respectively. As well, an application developer can assign the priorities based on the criteria such as interactive tasks first, foreground jobs first, likewise to fulfill the application requirement. This change in the order of tasks' execution also benefits the overall system performance by giving reduced AWT and reduced ATT resulting inefficient utilization of limited resources and better throughput of the overall system.

REFERENCES

[1] AkhilendraPratap Singh, Ashish Kr Luhach, Xiao-ZhiGao, Sandeep Kumar and DiptenduSinha Roy. Evolution of wireless sensor network design from technology-centric to user-centric: An architectural perspective, International Journal of Distributed Sensor Networks, 16(8) (2020) DOI: 10.1177/1550147720949138

[2] Ahmad Ali [1,*], Yu Ming [1], SagnikChakraborty[2] and SaimaIram[2]. Review-A Comprehensive Survey on Real-Time Applications of WSN, Future Internet, 9(77) (2017) ; doi:10.3390/fi9040077 www.mdpi.com/journal/futureinternet

[3] DionisisKandris, Christos Nakas, DimitriosVomvas and GrigoriosKoulouras.Applications of Wireless Sensor Networks: An Up-to-Date Survey- Review. Appl. Syst. Innov. 3(14)(2020) doi:10.3390/asi3010014 www.mdpi.com/journal/asi

[4] Rebin B Khoshnaw[1], Dana FarhadDoghramachi, Mazin S. Al-Hakeem.A Review on Internet of Things Operating Systems, Platforms, and Applications. Conference Paper · DOI:10.23918/iec2017.06 February (2017)

[5] ZoranCekerevac, Zdenek Dvorak, Tamara Pecnik.Top seven IoT operating systems in mid-2020. MEST Journal DOI 10.12709/mest.08.08.02.06 Published: July 8(2) (2020) 47-68

[6] AdiMallikarjuna Reddy V AVU Phani Kumar, D Janakiram, and G Ashok Kumar.Operating Systems for Wireless Sensor Networks: A Survey Technical Report. May 3, (2007) 1-30

[7] Nahla S. Abdel Azeem 1, Ibrahim Tarrad 2, Anar Abdel Hady 3,4, M. I. Youssef 2, and Sherine M. Abd El-kader 3,*.Shared Sensor Networks Fundamentals, Challenges, Opportunities, Virtualization Techniques, Comparative Analysis, Novel Architecture, and Taxonomy. Journal of Sensor and Actuator Networks. (2019), doi:10.3390/jsan8020029

[8] Muhammad Amjad, Muhammad Sharif, Muhammad Khalil Afzal, and Sung Won Kim.TinyOS-New Trends, ComparativeViews, and Supported Sensing Applications: A Review. IEEE SENSORS JOURNAL, 16(9) (2016).

[9] Hugo Landaluce, Laura Arjona, AsierPerallos, Francisco Falcone, Ignacio Angulo and Florian Muralter.A Review of IoT Sensing Applications and Challenges Using RFID and Wireless Sensor Networks. Sensors 2020, 20, 2495; doi: 10.3390/s20092495 available at www.mdpi.com/journal/sensors

[10] Michael Healy, Thomas Newe, ElfedLewis.Power Management in Operating Systems for Wireless Sensor Nodes. SAS 2007 - IEEE Sensors Applications Symposium San Diego, California USA, (2007) 6-8.

[11] DolvaraGunatilak -Based on tutorial by Mo Sha, RahavDor.TinyOS Tutorial. CSE521S, Spring 2017, CPSL, Cyber-Physical System Laboratory. (2017)

[12] Yousaf Bin Zikria 1, Sung Won Kim 1,*, Oliver Hahm 2, Muhammad Khalil Afzal 3, and Mohammed Y. Aalsalem 4. Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution. Sensors, 19(2019) 1793; doi:10.3390/s19081793 www.mdpi.com/journal/sensors

[13] Dang Huynh-Van, Ngan Le-Thi-Chau, Khoa Ngo-Khanh, Quan LE-TRUNG.Towards an Integration of AES Cryptography into Deluge Dissemination Protocol for Securing IoTs Reconfiguration. 2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF).

[14] Cíntia B. Margi, Bruno T. de Oliveira, Gustavo T. de Sousa, Marcos A. SimplicioJr, Paulo S. L. M. Barreto, Tereza C. M. B. Carvalho, Mats Näslund, Richard Gold,Ericsson.Impact of Operating Systems on Wireless Sensor Networks (Security) Applications and Testbeds. IEEE, 978-1-4244-7116-4/10/$26.00 ©2010 IEEE 2010

[15] Piergiuseppe Di Marco, Ericsson Research, Lecture 2 Introduction to Programming WSNs. Principles of Wireless Sensor Networks, https://www.kth.se/social/course/EL2745/, September 1, 2015

[16] Martin Perner.TinyOS Part 1. 182.694 Microcontroller VU, SS 2017

[17] Martin Perner.TinyOS Part 2. 182.694 Microcontroller VU, SS 2017

[18] Anita Patil (1), Dr. Rajashree.V.Biradar (2).Scheduling Techniques for TinyOS: A Review. International Conference on Computational Systems and Information Systems for Sustainable Solutions, 978-1-5090-1022-6/16/$31.00 ©2016 IEEE, (2016) 188-193.

[19] Elhadi M. Shakshuki, Stephen Isiuwe. Resource Management Approach to an Efficient Wireless Sensor Network. 9th International Conference on Emerging Ubiquitous Systems and Pervasive NetworksEUSPN 2018 Procedia Computer Science 141 (2018) 190–198.

[20] Salahuddin M. ElKazak, Cairo University, Masters in Computer Engineering.GEN600 Final Technical Report: Research in the Internet of Things Operating Systems (IoT OS's). Research in IoT OS's, GEN600: Final Technical Report

[21] TinyOS homepage(accessed 2021) available at : http://tinyos.stanford.edu/tinyos-wiki/index.php/MSPSim

[22] Anita Patil, Dr.Rajashree.V.Biradar.Programming the Sensor Nodes in WSN. International Journal of Engineering and Advanced Technology (IJEAT), ISSN: 2249–8958, Volume-8, Issue-2S, December 2018

[23] Wikipedia information about Sensor nodes (accessed 2021), available at https://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes

[24] Ram Prasadh Narayanan[1*], ThazathVeedu Sarath[2], VelloraVeetil Vineeth[3].Survey on Motes Used in Wireless Sensor Networks: Performance & Parametric Analysis. Wireless Sensor Network, 2016, 8, 51-60 Published Online April 2016 in SciRes.http://www.scirp.org/journal/wsn. 2016

[25] Walter Tiberti, DajanaCassioli, Antinisca Di Marco, Luigi Pomante and Marco Santic.A Model-Based Approach for Adaptable Middleware Evolution in WSN Platforms. Journal of Sensor and Actuator Networks. 2021, doi : 10.3390/jsan10010020 https://www.mdpi.com/journal/jsan

[26] Wikipedia information about NesC (accessed 2021), available at https://en.wikipedia.org/wiki/NesC

[27] TinyOS supporting document (accessed 2021), available at https://github.com

[28] TinyOS homepage (accessed 2021) for TEPs, available at: http://tinyos.stanford.edu/tinyos-wiki/index.php/TEPs

[29] Naji A. Majedkan[1,*], Abdulraheem J. Ahmed[2], Lailan M. Haji[3], (2020).CPU Scheduling Techniques: A Review on Novel Approaches Strategy and Performance Assessment. Journal of Applied Science and Technology Trends 01(02) (2020) 48 –55, doi: 10.38094/jastt1215.