EAI Endorsed Transactions
on Creative Technologies
Research Article EAI.EU

An Investigation on Several Operating Systems for Internet of Things

Fawwad Hassan Jaskani[1,*], Saba Manzoor[2], Muhammad Talha Amin[3], Muhammad Asif[2] and Muntaha Irfan[4]

[1]Khawaja Fareed University of Engineering and Information Technology, Rahim Yar Khan
[2]Islamia University of Bahawalpur, Bahawalpur
[3]University of Lahore, Lahore
[4]National University of Science and Technology, Islamabad

Abstract

In the field of development, Internet of things (IoT) plays a crucial role in providing solution to various situations. A lot of research has been conducted recently to model IoT based operating systems as standard UNIX, Windows and current real time operating systems are unable to meet the demand of heterogeneous IoT applications. In this paper we will focus on major OS features such as architecture, programming model, portability, memory management, real-time environment, scheduling algorithm, hardware support, networking and energy efficiency. We will be focusing on the following six operating systems which are as follows: Contiki, Tiny OS, RIOT, Zypher, Mbed and Brillo.

Keywords: Internet of things, Operating Systems, Contiki, Tiny OS, RIOT, Brillo, Zephyr.

Received on 30 October 2018, accepted on 19 December 2018, published on 30 January 2019

doi: 10.4108/eai.13-7-2018.160386

1. Introduction

The Internet of Things (IoT) turns out to be an essential feature in human life. The IoT devices are very dynamic and heterogeneous in nature being used in smart vehicles, remote sensing, smart car parking, smart phones, electronic appliances, controlling and monitoring systems etc. IOT integrated system provides broad services in a connected network for exchanging information.

IoT works in embedded systems, which has wireless sensors to enable connection and communication over a network. IoT provides comprehensive data exchange facilities in a linked network [1-2]. Normally, IoT devices have less memory and energy resources. OS functions as a resource manager that manages certain resources such as CPU time, secondary storage such as hard disk, memory and network throughput. An IoT OS is intended to operate within the limitations of Internet of Things that include size, memory, energy and processing capability [3].

These discrete features of IoT are required for portable, efficient, flexible and light-weight system with small memory tracks. Various operating systems such as Windows 8.1, ARM, and Linux etc. are in a serious competition to design IoT based Operating Systems. The primary goal of this paper is to provide a comparative study amongst the terms which include architectural design, scheduling, programming language model, memory management and probability along with hardware support and a few inclusive drawbacks [4]. Figure 1 shows the layout of different IoT operating systems which have been discussed in this paper. Table 1 depicts the comparison of

*Corresponding author. Email:Favadhassanjaskani@gmail.com

IoT operating systems. Rest of the paper is presented as follows. Section II discusses the related work of different researchers who have proposed different IoT based Operating Systems. Section III discusses problem statement. In Section IV different parameters are discussed for IoT operating Systems. Section V highlights the Operating System for IoT devices. Finally, Section VI concludes the whole research and provides the future direction in this area.

2. Literature Review

Many research studies have focused on the operating system for IOT devices. Researchers have suggested various IOT operating systems for different IOT devices. They have conducted an experiment to study some features of IoT which are carried out in the way of development of ideal Operating System for IoT. Various researches have been done to experiment the features and strategies to be adopted by Operating systems to control the functioning of IoT systems. [3]

Gaur and et al [4] have suggested general structure for an IoT OS and provided a comparison of various existing operating system like Contiki, RIOT, Tiny OS, Lite OS, Free RTOS, Mantis OS, NutOS, SOS, Erika Enterprise, OpenTag, uClinux, Nano-RK. He has concluded that there are some special features of IoT devices as compared to typical computer base devices. Therefore, the OS should be designed in such a way that it must fulfil the requirements of IoT base devices in the required application area. Sabri and et al [2] have proposed different OS and decided according to their usage and newness in the domain of IoT for the past two years.

The chosen OSs such as Contiki OS and Tiny OS are most recognized event driven model and Mbed OS, FreeRTOS and RIOT Operating System as Real Time Operating Systems. In his review he suggested seven significant features for an IoT Operating System like architectural design, scheduler, memory footprint, programming language model, real time capabilities, hardware support, energy efficiency and network connectivity. In [7] several operating systems were compared. This work is different from other in that the selection of Operating Systems is comprised on the lasts three year's surveys.

Many questions were asked about the most used IoT operating system. The answer is not clear for some reasons. First, OS for an IoT is a latest area of research however the concept is old. Second, the Operating Systems are limited and new because the discipline is new and the third reason is that the experts for IoT are very small in number and most researchers, developers and even some organizations are in the starting phase of learning.

Mike and et al [28] provides an assessment on the IoT devices suggesting their attributes with their appropriate restrictions and use cases. There are multiple contributions. First, the idea of the IoT devices and its classification has been given with the aim to study different devices on the Internet of Things. Second, fundamental issues in the design of IoT devices have been reviewed. Third, an extensive survey of latest embedded devices and boards are carried out concentrating on main characteristics such as processing and memory capacities, safety characteristics, communication interfaces, size, price, OS assistance, energy requirements, battery life and projects for each device.

Chandra and et al [1] presents expositive review material on operating systems presently accessible for most emerging field IoT. Musaddiq and et al [12] has made an effort to provide insight into the IoT Operating System Resource Management Area of various proposed approaches. His article offers the features of various IoT OS protocols, their design procedures and their suitable benefits and constraints.

In [14], the sensor network's Tiny OS is presented. It described that Tiny OS is a completed system; it keeps evolving and growing. It is very promising to use language instruments for system-wide optimization. Components follow implicit software protocols.

In [13], Contiki OS is described. It is a lightweight, open source operating system developed for WSN. Then an instance scenario was clarified step by step through Cooja in Contiki OS and lastly a comparison was made with other common operating systems like TinyOS and LiteOS. If we look at Contiki OS, it is evident that it has strong tools to build complex wireless communication systems. Rime Network Stack is particularly important because it features a lightweight network stack that is very convenient for low-powered WSNs.

In [10], LiteShell subsystem is discussed that offers sensor nodes with a command line interface such as UNIX. This shell runs on the base station's PC side. Consequently, it is a front-end interacting with the user. In [8], RIOT OS is suggested. It seeks to bridge the gap that has been observed between WSN operating systems and traditional full-fledged operating systems presently operating on internet hosts. It is comprised on design goals including low memory footprint, energy efficiency, modularity and uniform access to the API, regardless of the hardware underlying it.

IoT consists of large applications with a broad range of hardware systems relying entirely on distinct architectures. It is extremely difficult to design one operating system that fulfils all the requirements of IoT specific applications [2]. An operating system should be developed in such a way

that avoid redundant development and satisfy various requirements. But neither the conventional operating system for wireless sensor networks, nor any modern developed operating systems are efficient to satisfy all the needs of IoT devices. This research contributes a comparative analysis centred on various IOT operating systems and reveals the features of each operating system to have an enhanced range in IoT particular applications.

3. Parameters for Selection of Suitable IoT OS

An operating system (OS) is software that manages computer hardware resources. It runs other program and provide application software services. Generally, the OS consist of kernel, system shell and utility software. The kernel is the most important component of OS. It manages the operations of the computer and the hardware especially memory and CPU time. Antivirus software, backup software, and computer instruments are examples of utility programs. Below are the fundamental parameters and specifications separating an IoT Operating system? These parameters are:

3.1 Architecture

The essential part of an Operating System is the kernel. The organization of the kernel consists of the OS structure that impacts both the size of the application programs and the manner in which it provides services. Some familiar operating system architectures are monolithic, Modular, microkernel and layered. Monolithic has a single huge process. It runs solely in a single address space of memory and it does not have any specific structure [5]. In this type of architecture the services are applied separately and each service presents an interface for others. The cost of monolithic OS module is low therefore the system is difficult to understand and maintain. Another problem of monolithic kernel is that the code is too long and complex therefore it is difficult to configure and understand. Because this type OS architecture is unreliable so it's not a good choice for IOT devices.

The architecture of microkernel has a simple structure. Microkernel architecture has separate process which is known as server. Some servers operate in user-space while others operate in kernel space. A microkernel is the best choice for many embedded operating systems. It is due to the tiny size of kernel and small number of context switches. Due to minimum functionalities the kernel size is reduced significantly. Moreover, this type of architecture offers higher reliability, customization and ease of expansion. Since most of the OS features such as time and memory server are delivered via user-level servers.

The modular architecture is however much better than monolithic, because a single module failure does not result in a complete system crash. The layered architecture method is less modular th an the microkernel method. It is more stable and less complex than the monolithic kernel.

3.2 Programming Model

The programming model decides that how program can be modelled by an application developer. Typical programming models can be split into event-driven and multithreaded schemes for IoT operating systems. An external event like an interrupt must trigger each job in an event-driven scheme.

A multi-threaded programming system offers the chance to interact between the functions using an Inter Process Communication (IPC) and to perform each job in its own thread context. The programming language should be developed in such a way that programmers can use the system efficiently [2].The selection of programming model is influenced by many variables.

Particularly parallelism, hierarchy of memory and competition decide which model to use. The model of programming in turn impacts the efficiency and productivity of the scheme. Assembly language is the finest hardware interface option, but high-level language support is required to render it easy to create. However, it is difficult to provide high-level languages on restricted platforms.

3.3 Scheduling Policy

Scheduling strategy is the main factor that determines system performance. The scheduling algorithm depends on the latency (response time, turnaround time), performance, energy efficiency, real-time capacities, fairness and waiting time. Two kinds of schedulers are available i.e. pre-emptive and cooperative. Pre-emptive scheduler is the one that allocates CPU time to each task while in cooperative model different jobs take different CPU time [9]. Several applications exist with strict time limits due to the variety of IoT tasks. The scheduler should handle real-time tasks in order to fulfill the deadlines and to complete the activities within certain time limits. In addition the schedulers in IoT systems should be multitasking and energy efficient.

3.4 Memory footprint

Memory management offers an idea of managing memory allocation, de-allocation, caching, logical and physical

address mapping, memory security and virtual memory. As devices are limited in number so an OS must have low memory and processing requirements [13]. IoT devices typically provide a few kilobytes of memory, millions of times less than connected machines (smartphones, laptops, tablets etc.)The amount of memory management requirement relies on the type of application and the underlying platform support. The distribution of memory may be static or dynamic. Memory distribution is easier through static method, but dynamic strategy can provide flexibility in runtime memory acquisition.

3.5 Networking

Connectivity of internet is a basic condition for IoT devices. It should be possible for the IoT organizations to communicate with low power consumption. OS supports various protocols of connectivity, such as Wi-Fi, Ethernet, BLE, IEEE 802.15.4, etc. IoT is not appropriate for traditional TCP / IP stacks or WSN networking technologies. While the previous expert's fails to attain the objectives of less memory, less complexity, and low power, the latter requires intermediate proxies to allow various communication platforms to talk to end to end users. In addition, WSN protocols such as ZigBee, Bluetooth, Wavenis, Z-Wave etc. comply with the specific demands of smart systems, but do not meet IoT's broad-based communication criteria [9]. To allow effective seamless Internet communication, we need an open standard. The IoT stack must be flexible in order to be configured to satisfy the requirements of a broad spectrum of IoT apps with minimal modifications. In IoT schemes, support for Ipv6 is compulsory to have distinctive identities in huge networks.

3.6 Portability

OS separates applications from software specifics. OS is usually ported to separate hardware devices and board support package (BSP) interfaces in a conventional manner. The operating system should be easily connected to different hardware systems. The big range of hardware architectures should be supported. The IoT micro-controllers used variety between 8-bit and 32-bit.

The OS should be prepared to exploit the design that underlies it [27]. In addition, IoT is a future with a broad spectrum of apps in various fields. The OS should be adaptable to the application's specific requirements and provide sensible abstraction to the background information. In addition, IoT is a future with a broad spectrum of apps in various fields [22]. The OS should be adaptable to the application's specific requirements and provide sensible abstraction to the background information.

3.7 Energy Efficiency

Energy efficiency becomes crucial for battery-powered IoT systems and should be considered when developing an IoT OS. Most IoT systems are resource-bound in nature [4]. Therefore, battery or other constrained energy sources are used to operate it. Scenarios for IoT implementation are varied, difficult and sometimes very distant. Humongous IoT network size requires IoT OS to operate the IoT equipment for many years to be power efficient.

4. Operating system for IoT devices

This section of paper presents some operating system for IoT devices. These operating systems have been chosen on the basis of a variety of requirements including architecture, portability, hardware support, memory requirement, energy efficiency and other exciting features and characteristics.

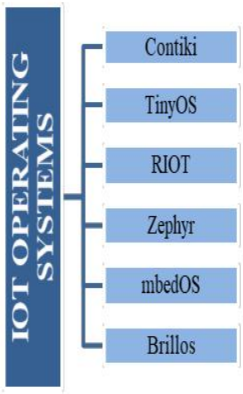

Figure 1. Layout of Different Operating Systems

4.1 Contiki

Contiki is an open source, flexible and lightweight IoT operating system. Open source implies the source is accessible and will be obtainable all the time. Contiki can be used without constraints in business and non-commercial applications. Contiki is implementing a model of hybrid protothread [11]. Protothread is a combination of event-driven programming systems and multi-threaded ones. It is fortified with strong, energy-efficient web communication facilities that connect low-cost, power-restricted small microcontrollers to the internet and run on various power-restricted appliances. Contiki uses the cooperative or preemptive based scheduling for the processes [15]. Contiki is intended to use well-known and well-tested IPv4, IPv6, and HTTP standards. The small memory demands create Contiki suitable for systems with low energy constraints. It is in C language. On daily basis

Contiki developers test the important aspect in Cooja simulator to ensure that Contiki code works as expected using nightly regression tests. Cooja is written in java and used as a single thread for simulation. Therefore, it cannot take advantage of multi-core processors and it requires a long time to complete the simulation for dense network situations [18]. It supports many IoT equipment such as wismote, sky and, z1.Contiki operates on a broad spectrum of small platforms, varying from 8051-powered -on-a-chip systems through the MSP430 and the AVR to a variety of ARM devices. Contiki offers a lot of options for support. Further, it needs to further develop to accommodate newly available IoT hardware platforms. Contiki operates on various hardware platforms and easily portable to new device. [19].

4.2 Tiny OS

Tiny OS is an open source and component based embedded operating system. Design of this OS includes low powered wireless system such as personal area networks (PANs), universal computing, smart buildings and smart meters. It can be used for further refinement for custom applications. It has been installed over a dozen platforms and various sensor boards [20]. It has been used by a large number of people to simulate, develop and test various algorithms and protocols of which is pretty clear from the number of downloads which is more than ten thousands [21]. Tiny OS version 2.1 has supportive TOS threads, i.e. if the CPU is not in use then it is the responsibility of an application to provide the CPU explicitly. Scheduler uses preemptive FIFO scheduling to execute threads and schedules as a high priority thread. It supports C programming language. As this is strongly related to the Tiny OS component-based model so that's why accessing hardware will become quite easy. During compile-time static memory allocation is used
[23]. Virtual memory ideas, feature pointers, heaps or primarily static memory distribution ideas do not exist.

4.3 RIOT

RIOT is free, an open source operating system. The operating system of RIOTS provides the developer with friendly environment. It enables in evolving own IoT's applications for compact devices on internet. RIOT contain few basic features such as multithreading, less power consumption, real time capabilities, reliability, small memory requirements, and constant API access. RIOT usually supports most microcontroller architectures (16-bit, 32-bit, 8-bit) and low-power IoT devices [25].

It purposes to implement the related open standards supporting an Internet of the Things which is secure, connected, privacy-friendly and durable. RIOT supports the programming languages of C++ and C. Also, it provides multithreading with preemptive, priority based and tickles scheduler. For reduction of the inherent drawbacks like code stack usage, inter-process messaging and thread management overhead, multithreading is designed [12]. Native is a hardware virtualizer or an emulator which allows a user to run RIOT code as Linux processes. Hence, it's easier developing IoT software without need of the actual hardware [27].

4.4 Zephyr

A small real-time operating system for resource-constrained, connected, embedded devices (with emphasis on the microcontrollers) and supports multiple architectures and was released under Apache License 2.0 is Zephyr. Zephyr includes complete essential libraries and components required in developing a complete application like protocol stacks, device drivers, firmware update and file systems, beyond its kernel. Zephyr, originally developed by the Intel subsidiary Wind River, provides microkernel for the less constrained devices of IoT and Nano kernel for the constrained devices.

It support multithreading with priority-based, cooperative, Earliest Deadline First (EDF), preemptive and non-preemptive scheduling. Programs can be coded in programming languages of C and C++. Zephyr provides with the network stack support consisting of multiple protocols [5]. Also, it supports the Bluetooth Low Energy (BLE) 5.0 [26]. Applications can be developed, build and tested by using the port of native posix. Zephyr contains no loadable kernel modules because the kernel is compiled statistically into a single binary file. This makes Zephyr safe from compile time attacks. Zephyr can run on a low memory devices. Interconnectivity technology is the major need of Zephyr which includes Wi-Fi, Bluetooth etc. ARM, ARC, RISC-V architectures are supported by zephyr [17].

4.5 Mbed OS

Mbed OS is an open-source embedded operating system. It is designed explicitly for the devices in the Internet of Things [24].Based on an on an Arm Cortex-M microcontroller we can produce a product that contain certain features such as connectivity, security and drivers for some sensors and I/O devices. The Real Time Operating System (RTOS) Mbed OS is created for restricted IoT systems by Advanced RISC Machine (ARM). It is designed specifically for ARM architecture of 32 bit [19]. It provides preemptive scheduler and is based on monolithic kernel. It supports development of C and C++. Low memory demands and different hardware support of Mbed OS make it appropriate for IoT based research. It includes a set of connectivity protocol stack drivers using, Cellular, Ethernet, ZigBee IP, Bluetooth, ZigBee NAN, Wi-Fi, etc. radio communication.

4.6 Brillo

Brillos an android based operating system is developed by google. It gives power to almost half of the world's smartphones. With at least 128 MB of ROM and 32 MB of RAM, it can operate on constrained/ low-end devices. The architecture of Brillo supports the processor of ARM, Intel and MIPS. It supports development in both C and C++ programming language. Built on top of the monolithic kernel, it offers a reasonable scheduler [3].

Memory requirements for Android things make it not suitable for IoT devices with low-end constraints rather than for high-end IoT devices. Lack of interoperable standards is one of the main problems of IoT devices. In such situations, Brillo is found to be appropriate. It provides a data synchronization protocol between devices called ' 'weave'.

The common standards communication layer of the Brillo is called Weave. Common language is used to communicate with sensors and devices. It can solve the fragmentation problem in home automation [27]. Brillo also offers device support to communicate over the mobile, allowing customers to control devices easily. Several intercommunication technologies such as Wi-Fi, Bluetooth are supported by Brillo

Table 1. Comparison of different IoT Operating Systems as referred in [10]

OS	Min RAM	Min ROM	C Support	C++ Support	Multi-threading	Architecture	Scheduler
Contiki	< 2 KB	<4 KB	~	x	~	Monolithic	Cooperative, Preemptive
Tiny OS	<1 KB	<30 KB	x	x	~	Monolithic	Cooperative
RIOT	~ 1.5 KB	~ 5KB	✓	✓	✓	Microkernel	Preemptive, Priority based
Zephyr	~ 2 KB to ~8 KB	~ 50 KB	✓	✓	✓	Microkernel	Preemptive, Priority based
Mbed	~ 5KB	~15 KB	✓	✓	✓	Monolithic	Preemptive
Brillos	~ 32KB	~128 MB	✓	✓	✓	Monolithic	Completely fair

5. Conclusion

The review presents and selects different Operating Systems according to their usage and newness in the field of IoT for the last three years. Compared to conventional computing systems, IoT has some unique characteristics [4]. The OS should therefore be designed according to the specific IoT device specifications and target application regions. In this paper, we conducted a survey of IoT's characteristics with an analysis of multiple attempts to develop the perfect IoT OS. There are many challenges that can motivate OS for IoT. Some challenges for kernel is that it should be light in weight, should have compatibility to handle Real-Time tasks and should have minimum energy and power consumption. This paper would assist researchers to know about Internet of Things, their traits of character, and OSes ' strategies for managing smart real-time IoT devices. There is no generic operating system that exists for IoT devices. We can choose the best operating system according to the requirement of IoT devices.

References

[1] T. Chandra, P. Verma and A. Dwivedi, "Operating Systems for Internet of Things", Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies - ICTCS '16,

[2] Y. Zikria, S. Kim, O. Hahm, M. Afzal and M. Aalsalem, "Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution", Sensors, vol. 19, no. 8, p. 1793, 2019.

[3] Y. Zikria, S. Kim, O. Hahm, M. Afzal and M. Aalsalem, "Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution", Sensors, vol. 19, no. 8, p. 1793, 2019.

[4] P. Gaur and M. Tahiliani, "Operating Systems for IoT Devices: A Critical Survey", 2015 IEEE Region 10 Symposium, 2015.

[5] M. Abdelsamea, M. Zorkany and N. Abdelkader, "Real Time Operating Systems for the Internet of Things, Vision, Architecture and Research Directions", 2016 World Symposium on Computer Applications & Research (WSCAR), 2016.

[6] A. Musaddiq, Y. Zikria, O. Hahm, H. Yu, A. Bashir and S. Kim, "A Survey on Resource Management in IoT Operating Systems", IEEE Access, vol. 6, pp. 8459-8482, 2018.

[7] M. Silva, A. Tavares, T. Gomes and S. Pinto, "ChamelIoT: An Agnostic Operating System Framework for Reconfigurable IoT Devices", IEEE Internet of Things Journal, vol. 6, no. 1, pp. 1291-1292, 2019.

[8] F. Javed, M. Afzal, M. Sharif and B. Kim, "Internet of Things (IoT) Operating Systems Support, Networking Technologies, Applications, and Challenges: A Comparative Review", IEEE Communications Surveys & Tutorials, vol. 20, no. 3, pp. 2062-2100, 2018.

[9] Z. Zhang, M. Cho, C. Wang, C. Hsu, C. Chen and S. Shieh, "IoT Security: Ongoing Challenges and Research Opportunities", 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications, 2014.

[10] V. Vanitha, V. Palanisamy, N. Johnson and G. Aravindhbabu, "LiteOS based Extended Service Oriented Architecture for Wireless Sensor Networks", International Journal of Computer and Electrical Engineering, pp. 432-436, 2016.

[11] L. Farkas and L. Losonczi, "Communication Protocol for Wireless Sensor Network, Dedicated for Real Time Biosignal Acquisition, Implemented on Top of Contiki OS", MACRo 2015, vol. 2, no. 1, pp. 105-111, 2017.

[12] I. Rasool, Y. Zikria, A. Musaddiq, F. Amin and S. Kim, "RIOT-OS: FIRMWARE FOR FUTURISTIC INTERNET OF THINGS", Far East Journal of Electronics and Communications, vol. 17, no. 4, pp. 877-887, 2017.

[13] R. Rodriguez-Zurrunero, R. Utrilla, A. Rozas and A. Araujo, "Process Management in IoT Operating Systems: Cross-Influence between Processing and Communication Tasks in End-Devices", Sensors, vol. 19, no. 4, p. 805, 2019.

[14] "Survey Paper for IOT, Capacity Planning and Cloud Technologies", International Journal of Science and Research (IJSR), vol. 5, no. 4, pp. 1812-1815, 2016.

[15] J. Gubbi, R. Buyya, S. Marusic and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions", Future Generation Computer Systems, vol. 29, no. 7, pp. 1645-1660, 2013.

[16] I. Skerrett, “IoT Developer Survey 2017,” LinkedIn SlideShare, 18-Apr-2017. [Online].

[17] “IoT Operating Systems,” Arrow.com, 31-Sep-2016.

[18] N. Windpassinger, “Internet of Things - the complete online guide to the IoT,” i-SCOOP. [Online].

[19] N. Windpassinger, “Internet of Things - the complete online guide to the IoT,” i-SCOOP. [Online].

[20] A. Elvstam and D. Nordahl, “Operating systems for resource constraint Internet of Things devices: An evaluation,” MUEP, 2016. [Online].

[21] M. Ojo, S. Giordano, G. Procissi and I. Seitanidis, "A Review of Low-End, Middle-End, and High-End Iot Devices", IEEE Access, vol. 6, pp. 70528-70554, 2018.