SOCIETY FOR SCIENCE AND EDUCATION
UNITED KINGDOM
TMLAI TRANSACTIONS ON MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE
VOLUME 5 NO 4, 2017
ISSN 2054-7390

A Comparative Study Between Operating Systems (Os) for the Internet of Things (IoT)

Aberbach Hicham, Adil Jeghal, Abdelouahed Sabrim, Hamid Tairi
LIIAN, Department of Mathematic & Computer Sciences, Sciences School, Sidi Mohammed Ben Abdellah University,
aberbachhicham@gmail.com, adil.jeghal@usmba.ac.ma, abdelouahed.sabri@gmail.com, htairi@yahoo.fr

ABSTRACT

Abstract : We describe The Internet of Things (IoT) as a network of physical objects or "things" embedded with electronics, software, sensors, and network connectivity, which enables these objects to collect and exchange data in real time with the outside world. It therefore assumes an operating system (OS) which is considered as an unavoidable point for good communication between all devices “objects”. For this purpose, this paper presents a comparative study between the popular known operating systems for internet of things . In a first step we will define in detail the advantages and disadvantages of each one , then another part of Interpretation is developed, in order to analyze the specific requirements that an OS should satisfy to be used and determine the most appropriate .This work will solve the problem of choice of operating system suitable for the Internet of things in order to incorporate it within our research team.

Keywords: Internet of things , network, physical object ,sensors,operating system.

1 Introduction

The Internet of Things (IoT) is the vision of interconnecting objects, users and entities “objects”.

Much, if not most, of the billions of intelligent devices on the Internet will be embedded systems equipped with an Operating Systems (OS) which is a system programs that manage computer resources whether tangible resources (like memory, storage, network, input/output etc.) or intangible resources (like running other computer programs as processes, providing logical ports for different network connections etc.), So it is the most important program that runs on a computer[1].

Every general-purpose computer must have an operating system to run other programs and applications.

Computer operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as printers. For large systems, the operating system has even greater responsibilities and powers. It is like a traffic cop — it makes sure that different programs and users running at the same time do not interfere with each other. The operating system is also responsible for security, ensuring that unauthorized users do not access the system, the most of these operating systems are used by the internet of things in several applications which are translate into many concrete uses - new or improved- significantly impacting the daily lives of individuals, businesses and communities for example[2] : Cities :

DOI: 10.14738/tmlai.54.3192
Publication Date: 15th August, 2017
URL: http://dx.doi.org/10.14738/tmlai.54.3192

the Internet of things will allow better management of the various networks that feed our cities (water, electricity, gas, etc.) by allowing continuous real-time and precise control, Energy : power grid management will be improved by telemetry, enabling real-time management of the energy distribution infrastructure ,Transport : in this field the Internet of Things will support the current efforts around intelligent vehicles for road safety and driving assistance ,Health : in the field of health, the Internet of things will allow the deployment of personal networks for the control and monitoring of clinical signs, especially for the elderly, The industry : in the Internet the Industry of Objects will allow a total tracking of the products, from production chain, up the supply chain and distribution by overseeing the supply conditions. Because the problem of choice between different operating systems presents an obstacle in our context. The main objective of this work is to solve it in a definitive way. This work contains a detailed study of the existing operating systems for the Internet of things, and this through an evaluation of each one with its advantages and disadvantages, and the final part will be devoted to defining and examine the appropriate Operating system to be used in the Internet of Things.

2 Operating Systems

The set of operating systems designed for the internet of things are presented under the architecture shown in Figure below (Figure1) : A hard-ware layer and drivers, a second layer that contains Kernel / scheduler and network stack, The choice of Kernel is to have a complete control over everything that occurs in the system and the choice of the scheduling strategy is tightly bound to real-time and different task priorities support .Finally the last layer is for applications or user interface for supporting degree of user interaction[3] :

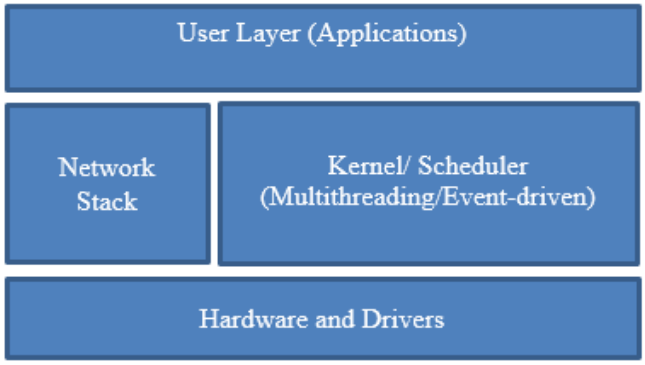

Figure1: architecture of operating systems

2.1 TinyOS

TinyOS [4] is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters.

2.1.1 Architecture:

TinyOS follow a monolithic architecture based on a combination of components, reducing the the size of the code needed to set it up. This is in line with the constraints of Memories that are observed by sensor networks; however, the TinyOS component library is particularly complete since it includes network protocols, sensor drivers and acquisition tools of data. All of these components can be used as they are, they can also be adapted to a precise application.

2.1.2 Programming Model:

The previous versions of TinyOS do not provide multithreading support; building applications were based on the event-driven programming model. TinyOS version 2.1 supports multithreading. and these TinyOS threads are called TOS Threads[5].

2.1.3 Suported plateforms:

TinyOSTinyOS supports the following sensing platforms: Mica [6], Mica2 [6], Micaz [6] and a few others.

2.1.4 Advantages

TinyOS is a system mainly developed and Supported by the American University of Berkeley, which offers it in download Under the BSD license and monitors it. Thus, all the sources are Available for many physical targets[7].

Based on event-driven operation, TinyOS offers to the user a precise management of the sensor's consumption and makes it possible to adapt better to the Random nature of wireless communication between physical interfaces.

Compatibility with at least 9 hardware platforms, and the simplicity of adding or modifying platforms. TinyOS follows a hierarchical abstraction of the material in three layers.

The preemptive nature of an operating system specifies if this one allows the interruption of a task in progress. TinyOS does not manage this Preemption between tasks but gives priority to hardware interruptions.

Energy management: TinyOS is designed to optimally manage energy consumption: it puts the node into standby when there are no tasks to perform, and thus allows the deactivation of the radio device or its setting in ' Listening only and low energy consumption (low power listening). TinyOS also generates a small code, which decreases the energy used to store the data in the RAM.

2.1.5 Disadvantages

a) Support for Real-Time Applications

TinyOS [7]does not support real-time application operations. It provides a process planning algorithm based on priorities, but once a process is scheduled to run to completion. This may result in a missed deadline for a high priority process that enters the Preparation queue once a low priority process has been scheduled

TinyOS is not an operating system in the current sense. It does not offer, for example, a notion of multitasking, users or file system. There are no user mode and kernel mode notions. In fact, TinyOS is a set of routines made available to the programmer to simplify the development.

2.2 Contiki

Contiki [8] is an open source operating systems for sensor nodes. It was developed at the Swedish Institute of Computer Science by Dunkels et al. [3], it is a lightweight open source OS written in C for WSN sensor nodes Contiki connects tiny lowcost, low-power microcontrollers to the Internet .

Contiki is a highly portable OS and it is build around an event-driven kernel. Contiki provides preemptive multitasking that can be used at the individual process level. A typical Contiki configuration consumes 2 kilobytes of RAM and 40 kilobytes of ROM. A full Contiki installation includes features like : multitasking kernel, preemptive multithreading, proto-threads, TCP/IP networking, IPv6, a Graphical User Interface, a web browser, a personal web server, a simple telnet client, a screensaver, and virtual network computing.

2.2.1 Advantages:

a) Internet Standards : Contiki provides powerful low-power Internet communication. Contiki supports fully standard IPv6 and IPv4, along with the recent low-power wireless standards: 6lowpan, RPL, CoAP. With Contiki's ContikiMAC and sleepy routers, even wireless routers can be battery-operated.

b) Rapid Development: With Contiki, development is easy and fast: Contiki applications are written in standard C, with the Cooja simulator Contiki networks can be emulated before burned into hardware, and Instant Contiki provides an entire development environment in a single download.

c) A Selection of Hardware : Contiki runs on a range of low-power wireless devices, many of which can be easily purchased online.

d) Open Source Software: Contiki is open source software: it can be freely used both in commercial and non-commercial systems and the full source code is available.

2.2.2 Architecture

Contiki is based on a modular architecture, with respect to its kernel that follows the event driven model, but it provides threading capabilities to processes. The Contiki kernel includes a light event scheduler that dispatches events to running processes. The execution of the process is triggered by events sent by the kernel to processes or by a query mechanism.

2.2.3 Disadvantages

a) Scheduling : Contiki is an event-driven OS, therefore it does not employ any sophisticated scheduling algorithm. Events are fired to the target application as they arrive.

Contiki does not support multicast. Therefore Contiki does not provide any implementation of group management protocols such as the Internet Group Management Protocol (ICMP), or Multicast Listener Discovery (MLD) protocol

b) Possibility of Real-Time support applications : Contiki does not support deployment of real-time applications, So there is no implementation of any real-time process scheduling algorithm in this OS.

2.3 Nano-RK

Nano-RK [9] is a fully preemptive reservation-based real-time operating system (RTOS) from Carnegie Mellon University with multi-hop networking support for use in wireless sensor networks.Nano-RK currently runs on the FireFly Sensor Networking Platform as well as the MicaZ motes. It includes a light-weight embedded resource kernel (RK) with rich functionality and timing support using less than 2KB of RAM and 18KB of ROM.

Nano-RK supports fixed-priority preemptive multitasking for ensuring that task deadlines are met, along with support for CPU, network, as well as, sensor and actuator reservations. Tasks can specify their resource demands and the operating system provides timely, guaranteed and controlled access to CPU

cycles and network packets. Together these resources form virtual energy reservations that allow the OS to enforce system and task level energy budgets[8].

2.3.1 Advantages

One of the goals for Nano-RK[9] was to facilitate application developers by allowing them to work in a familiar multitasking paradigm. This results in a short learning curve, rapid application development, and improved productivity

a) Resource Sharing: For shared resources such as memory, Nano-RK provides semaphores for serialized access. To circumvent the priority inversion problem, Nano-RK provides an implementation of the Priority Ceiling algorithm. In addition, Nano-RK provides APIs to reserve system resources like CPU cycles, sensors, and network bandwidth.

b) Possibility of Real-time Applications support : Nano-RK is a real-time operating system; hence it provides rich support for real-time applications. It supports real-time processes and its offline admission control procedure guarantees to meet deadline associated with each admitted real-time process.

Nano-RK provides an implementation of real-time preemptive scheduling algorithms and tasks are scheduled using a rate monotonic scheduling algorithm. Moreover, Nano-RK provides bandwidth reservations for delay-sensitive flows and it claims to provide end-to-end delay guarantees in multi-hop wireless sensor network.

Nano-RK is a suitable OS for use in multimedia sensor networks due to its extensive support provided to real-time applications.

2.3.2 Disadvantages

Nano-RK only provides support for static memory management; it does not support dynamic memory management. In Nano-RK, both the OS and applications reside in a single address space and to the best of authors' knowledge Nano-RK does not provide any support to safeguard co-located OS and process address spaces.

Nano-RK is a preemptive multitasking OS, it needs to save the context of the current task before scheduling the new task. Saving the state of each task results in large memory consumption and frequent context switches result in reduced performance and higher energy consumption.

2.4 LiteOS

LiteOS [10] is an open source, interactive, UNIX-like operating system designed for wireless sensor networks. With the tools that come with LiteOS, you can operate one or more wireless sensor networks in a Unix-like manner, transferring data, installing programs, retrieving results, or configuring sensors. You can also develop programs for nodes, and wirelessly distribute such programs to sensor nodes.

2.4.1 Architecture

LiteOS [10] follows a modular architecture design. LiteOS is partitioned into three subsystems: LiteShell, LiteFS, and the Kernel. LiteShell is a Unix-like shell that provides support for shell commands for file management, process management, debugging, and devices.

2.4.2 Advantages

a) Scheduling

LiteOS provides an implementation of Round Robin scheduling and Priority-based scheduling. Whenever a task is added to the ready queue, the next task to be executed is chosen through priority-based scheduling. The tasks run to completion or until they request a resource that is not currently available. When a task requires a resource that is not available, the task enables interrupts and goes to sleep mode.

b) Memory Protection and Management

Inside the kernel, LiteOS supports dynamic memory allocation through the use of free functions. User applications can use these APIs to allocate and de-allocate memory at run-time. Dynamic memory grows in the opposite direction of the LiteOS stack. The dynamic memory is allocated from the unused area between the end of the kernel variables and the start of the user application memory blocks. This allows adjusting the size of dynamic memory as required by the application.

c) Communication Protocol Support

liteOS provides communication support in the form of files. LiteOS creates a file corresponding to each device on the sensor node. Similarly, it creates a file corresponding to the radio interface. Whenever there is some data that needs to be sent, the data is placed into the radio file and is afterward wirelessly transmitted. In the same manner, whenever some data arrives at the node it is placed in the radio file and is delivered to the corresponding application using the port number present in the data .

2.4.3 Disadvantages

a) Possibility of Real-time Applications support

LiteOS does not provide any implementation of networking protocols that support real-time multimedia applications. It provides a priority-based process scheduling algorithm but once a process is scheduled it runs to completion.This can result in a missed deadline of a higher priority process that enters the ready queue once a low priority process has been scheduled.

The LiteOS documentation does not provide enough detail on how system resources are shared among multiple executing threads.

2.5 FreeRTOS

“FreeRTOS [11] is a market leading RTOS from Real Time Engineers Ltd. that supports 35 architectures and received more than 113000 download during 2014. It is professionally developed, strictly quality controlled, robust, supported, and free to embed in commercial products without any requirement to expose your proprietary source code. FreeRTOS has become the de facto standard RTOS for microcontrollers by removing common objections to using free software, and in so doing, providing a truly compelling free software model.”

2.5.1 Advantages [11]

- FreeRTOS is downloaded every 260 seconds (on average).
- FreeRTOS offers lower project risks and a lower total cost of ownership than commercial alternatives because:
- It is fully supported and documented.

- Most people take products to market without ever contacting us, but with the complete peace of mind that they could opt to switch to a fully indemnified commercial license (with dedicated support) at any time.
- Some FreeRTOS ports never completely disable interrupts.
- For strict quality control purposes, and to remove all IP ownership ambiguity, official FreeRTOS code is separated from community contributions.
- FreeRTOS has a tick-less mode to directly support low power applications.
- FreeRTOS was downloaded >113000 times in 2014.
- FreeRTOS is designed to be simple and easy to use: Only 3 source files that are common to all RTOS ports, and one microcontroller specific source file are required, and its API is designed to be simple and intuitive.

2.5.2 Disadvantages

a) Limited Tasks

There are only limited tasks run at the same time and the concentration of these system are on few application to avoid errors and other task have to wait. Sometime there is no time limit of how much the waiting tasks have to wait.

b) Use heavy system resources

FreeRTOS used lot of system resources which is not as good and is also expensive.

c) Low multi-tasking

Multi-tasking is done few of times and this is the main disadvantage of FreeRTOS because this system runs few tasks and stay focused on them. So it is not best for systems which use lot of multi-threading because of poor thread priority.

d) Complex algorithms

FreeRTOS uses complex algorithms to achieve a desired output and it is very difficult to write that algorithms for a designer.

e) Device driver and interrupt signals

FreeRTOS must need specific device drivers and interrupt signals to response fast to interrupts.

f) Thread Priority

Thread priority is not good as FreeRTOS do less switching of tasks.

2.6 RIOT

RIOT [12] or the friendly Operating System for the Internet of Things is a lightweight operating system for networked systems with memory constraints, focused on devices with low power consumption for Internet of things. It is a free software, released under the GNU General Public License (LGPL)[5] , It was originally developed by the Free University of Berlin, the National Institute for Research in Computer Science and Automation (INRIA) and the University of Applied Sciences of Hamburg (HAW Hamburg). The core of RIOT is largely inherited from FireKernel1, which was originally developed for sensor networks.

2.6.1 Architecture

The microkernel architecture written in ANSI C and support for full multithreading enables a developer-friendly API. POSIX compliance is partly already available and full POSIX compliance is planned for the near future. Since RIOT is completely written in C, it also allows for the usage of C++ and the utilization of the GNU Compiler Collection (GCC) in the latest version.

2.6.2 Modularity

To ensure minimal memory usage, the system is based in a modular way. Thus, the configuration of the system can be customized to meet the particular specification. The size of the kernel itself is minimized, thus requiring only a few hundred bytes of RAM and program storage. Dependencies between the modules are reduced to an absolute minimum .

2.6.3 Advantages

RIOT [12] an operating system designed for the particular requirements of Internet of Things (IoT) scenarios. These requirements comprise a low memory footprint, high energy efficiency, real-time capabilities, a modular and configurable communication stack, and support for a wide range of lowpower devices.

RIOT provides a microkernel, utilities like cryptographic libraries, data structures (bloom filters, hash tables, priority queues), or a shell, different network stacks, and support for various microcontrollers, radio drivers, sensors, and configurations for entire platforms. There are no new programming environments.,C or C++ can be used directly with existing tools like gcc, gdb, etc, Less hardware dependent code, Supports 8-,16- and 32-bit microcontroller platforms, Energy efficiency is maintained, Less interrupt latency, so real-time capability is ensured, Multi-threading is enabled .

2.6.4 Network Stack

Supports the entire network stack of IoT (802.15.4 Zigbee, 6LoWPAN, ICMP6, Ipv6, RPL, CoAP, etc) , Both static and dynamic memory allocation, POSIX compliant (partial), All output can be seen in the terminal if hardware is not available.

RIOT is of course free and its code is available online. It is a fundamental prerequisite for developing a robust and sustainable technology and protocols on a global scale. Open source makes it possible to constantly improve a program due to a large community of contributors.

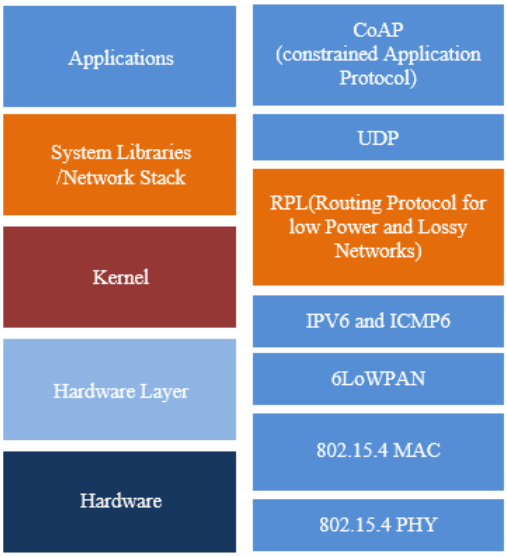

Figure 2: architecture and Driver support for RIOT

Table 1: comparison of OS's

OS	*License*	*Programming Model*	*Real-time support*	*Technologies supported*
TinyOS	***Open Source BSD***	***Event Drive,thread***	***No***	Broadcast based Routing
Contiki	***Open Source BSD***	***Protothreads and events***	***Partial***	6lowpan, RPL
NanoRK	***Open Source***	***Multithreading***	***Yes***	RTLink,U-Connect
LiteOS	***Open Source***	***Threads and Events***	***Partial***	JTAG
Free RTOS	***Open Source***	***Multithreading***	***Yes***	TCP,UDP, Ethernet, TLS
RIOT	***Open Source LGPL v2.1***	***Multithreading***	***Yes***	6LoWPAN, RPL,CoAP, UDP,TCP, CBOR, CCN-lite, OpenWSN, UBJSON

3 Comparison of IOT OS'S

As we can see in the following table (table1), the RIOT system supports the most communication technologies over other systems, supports multi-threading and real-time media with a GNU GPL license, which allows [13] the user several rights On a computer program.

1. The freedom to run the software for any purpose.
2. The freedom to study the functioning of a program and to adapt it to its needs, which requires access to source codes.
3. Freedom to redistribute copies.
4. The obligation to provide the community with modified versions.

Since Our goal is to discover and compare in a detailed way the characteristics of each operating system in order to detect the most suitable for the Internet of things. For this we have based on the characteristics

detailed in section2: OSs to conclude that the RIOT operating system is the most appropriate especially in our search axis since it is free to be run in any domain, adaptable To the needs of the user with simple access to source codes, it aims exactly to fulfill the requirements to should satisfy an operating system for internet of things .

4 Requirements of an OS for IOT

In summary the main requirements for an operating system are:

- Memory Requirements: To ensure proper memory management for an operating system the minimum memory requirement of the software must be very low. This concerns RAM as well as persistent program storage.
- Limited resources: The hardware platforms offer very limited resources so the operating system should use them eciently.
- Concurrency: The operating system should be able to handle diferent tasks at the same time.
- Flexibility: Since the requirements for deferent applications vary wildly, the operating system should be able to be flexible to handle those.
- Low Power: Energy conservation should be one of the main goals for the operating system.
- Error Free: Error free that mean it has no chances of error in performing tasks.
- Platform support: The software part of the IOT should have the ability to support the different hardware platforms, but also the ability to exploit their capabilities.
- Reliability: The operating system must function reliably because it is deployed in critical applications where the access is linked to a high costs

5 Conclusion and Future Work

In this work we have defined the notion of the Internet of things and its different fields of applications.

We have also compared the current (IoT) operating systems in a detailed way to see their advantages and disadvantages as well as their characteristics and architectures. Through this comparison we have deduced that the RIOT is the most appropriate system for the Internet of things since most of its Strengths correspond exactly to the Requirements that an operating system must satisfy to be used in the internet of things .So this article helps to better choose which operating systems to adapt for the internet of the things .This paper analyzed these parts in a single conclusion: RIOT is an operating system designed for the particular requirements of Internet of Things (IoT) Scenarios.

The future work well includes complete and compliant implementations of RIOT, on the most known platforms for the IoT.

RÉFERENCES

[1] ITU-T (2012). Overview of the Internet of things. ITU-T. Retrieved from http://www.itu.int/rec/T-REC-Y.2060-201206-I on 2016, Jan.

[2] Cisco IBSG © 2011 Cisco et/ou ses filiales. Tous droits réservés.

[3] Padmini Gaur, Mohit P. Tahiliani .Wireless Information Networking Group (WiNG)

[4] A. Dunkels, B. Gr¨onvall, T. Voigt Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors In Proceedings of the First IEEE Workshop on Embedded Networked Sensors, Tampa, Florida, USA, 2004

[5] Klues K, Liang CJM, Paek J, Musaloiu R, Levis P, Terzis A, Govindan R. TOSThread: Thread-Safe and Non-Invasive Preemption in TinyOS. Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems; Berkeley, CA, USA. 4–6 November 2009; pp. 127–140.

[6] PtolemyProject. Availableonline http://ptolemy.eecs.berkeley.edu/ (accessed on 17 April 2011)

[7] Akyildiz IF, Melodia T, Chowdhury KR. A Survey on Wireless Multimedia Sensor Networks. Comput. Netw. 2007;51:921–960

[8] Dunkels A, Gronvall B, Voigt T. Contiki a Lightweight and Flexible Operating System for Tiny Networked Sensors. Proceedings of the 9th Annual IEEE International Conference on Local Computer Networks; Washington, DC, USA. October 2004; pp. 455–462.

[9] Eswaran A, Rowe A, Rajkumar R. Nano-RK: An Energy-Aware Resource-Centric RTOS for Sensor Networks. Proceedings of the 26th IEEE Real-Time Systems Symposium; Miami, FL, USA. 5–8 December 2005.

[10] Cao Q, Abdelzaher T, Stankovic J, He T. The LiteOS Operating System: Towards Unix Like Abstraction for Wireless Sensor Networks. Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008); St Louis, MO, USA. 22–24 April 2008.

[11] Julien Le Sech, thirsday 5 january 2012. FreeRTOS sur ATmega328 .

[12] RIOT (2015). RIOT - The friendly Operating System for the Internet of Things. RIOT-OS.org. Retrieved from www.riot-os.org/ on 2016, Jan.

[13] Free Software Foundation https://www.gnu.org/licenses/lgpl-3.0.en.html .