Survey on Operating Systems in IoT Platform

Keerthi Naredla

Abstract- ***The idea of ubiquitous computing paved way for the internet of things which is now one of the fastest growing technologies. In the evolution of internet era, a simple device in this web of connected devices comes with sensors, processor, connections to other devices on platform or cloud services and even sometimes user-interface such as speech-recognition, display. These increasing features leads to the necessity of designing an operating system for IoT platform that meets requirements of the device to survive in the huge network of things. For example, turning on the air conditioner before you arrive home remotely. There are limitless possibilities and applications in this technologically growing world for IOT. As a result, developers introduced new variety of IoT OS over time. In this paper, I will be discussing 5 famous Operating Systems on IOT platform their advantages and challenges with the intent to show consistent improvements in Operating System in IoT platform.***

Keyword: *IoT, Wireless Sensor, Operating System, TinyOS, Contiki, Android Things,Ubuntu Core*

Introduction:

A simple device in the internet of things comes with sensors, processor, connections to other devices on platform or cloud services and user-interface such as speech-recognition, display. Clearly, with increasing complexity these systems require more than 16 KB ROM, and 32-bit micro-controller units, which in turn demands an Operating System to take care of underlying low-level challenges such as memory, process management etc. However, a usual OS in mobile phone, laptops and other devices cannot serve the purpose, due to limited resource and hardware constraints of the device. Here comes into the picture Embedded Operating System, which is a clearly important choice to make as it plays the major role in either limiting or enhancing the capabilities and flexibility of IoT implementations. It is also necessary to assess the future needs along with the current needs of the IOT system. Both open source and commercial OS can be used based on the need of the application and skills of the implementation team.

Keeping this in mind many developers introduced new variety of Embedded Operating Systems for Iot of devices over time. From embedded to specific all Operating System in Iot.

In this paper, I will be discussing 5 famous Open-Source Operating System on IOT platform their advantages and challenges with the intent to show the line of evolution of Operating System in IOT.

In this paper, we will first see the general requirements of an Operating System to serve in IOT platform, based on which we further explore different Operating Systems and determine how well they satisfy these requirements. In section 2, we explore OS on IOT which are designed to

serve in wireless network alone. In section 3, we will discuss more advanced OS in IOT which can be served in multiple fields. The paper closes in section 4, with the conclusion and future work.

General Requirements

Unlike personal computers, iot's are connected to a large number of sensors (they sense and transmit data) and gateways which connects to the cloud platform. Operating system chosen for iot implementation depends upon the context they are used, functions of sensors and their forms and their target applications. So, the main characteristics and requirements of operating system chosen for IOT implementations are as below,

1.**Energy efficiency**: It is important for OS to operate as highly efficient regarding energy because most of them use micro controllers or sensors that are battery powered or draws low power but furthermore expensive to replace batteries etc.

2.**Small memory footprint**: One of the restriction would small memory because usually, sensor nodes are small in size and stores limited memory

3. **Real-time capabilities**: In real-time applications, IOT has to perform similar to the industrial grade as real-time devices like medical, security-related systems needs to be accurate with time and critical in nature.

4**.Security:** The chosen OS has to adhere to the strict security rules and regulations that are imposed in critical environments.

5.**Network Connectivity and Protocol Support**: Connection to network and devices in near by proximity is important in device operation and also the variety of support needed for connecting to Bluetooth, wifi etc. For all this requirement, the operating system should help in simplifying the connectivity process.

6.**Hardware operation:** It is important for OS to support a variety of hardware platforms to simplify interconnectivity, drive standardizations and low ownership costs.

TinyOS

TinyOS was created at the University of California, Berkeley in 1999, and later maintained by many companies and universities. It is an Open Source Software, designed for low-power wireless embedded systems.

TinyOS is written in a dialect of C called NesC, which is the basis for major advantages of this Operating System. To understand it's benefits, let us first see how it supports the working of TinyOS.

TinyOS is components based OS. Each of the components has 3 interfaces commands, events, and tasks. A command is a request and upon its completion, an event signal is generated. Whereas Task is also a request for the component that is scheduled to execute later on.

A TinyOS consists of many components and interaction between each of these components takes place via an interface. These interfaces can be bidirectional, allowing complex interactions between components.

To make these interactions more feasible TinyOS implements split-phase execution model which prevents a component from waiting to receive an event signal for a command request. Both hardware and software modules of TinyOS implements this model, which increases the performance of TinyOS.

The concept of Tasks in TinyOS enable concurrency as the Scheduler executes each task in FIFO order and this, in turn, saves memory because each task can use the same stack. If there are no tasks in the task queues, the scheduler puts the CPU in sleep mode, which saves the energy-consumption of CPU. Moreover, TinyOS has a technique to save the energy of the components.

Although all tasks in TinyOS are non-preemptive, an interrupt can preempt tasks. this may lead to race conditions that imply concurrency issues arise. NecS prevents this by detecting the possible data races at compile time and warns the programmer. Also, NecS compiler reduces the processing power by eliminating boundary crossing between different components and considering as one whole program. Also, it does significant code optimization, as a result, reduces the program memory

When it comes to network protocol stack, TinyOS has set of API's for common functionality such as sending packets, receiving packets across the internet. It also supports 6LowPAN implementation which can be implemented over UDP.

However, TinyOS does not provide any specific MAC, network, or transport layers protocol implementations. Hence it cannot provide any explicit support for real-time applications. Not only this, TinyOS supports only certain hardware platforms and it is hard to port on a hardware platform that is not supported by TinOS. These constraints are overcome by our next OS Contiki.

Contiki

Contiki was originally created by Adam Dunkels in 2002 at the Swedish Institute of Computer Science. As TinyOS, it is an open source embedded operating system, designed for networked and memory constrained systems.

Contiki is based on a modular kernel that supports dynamic loading and linking, as a result, software modules are loaded only when the system requires their particular functionality. To achieve this, Contiki offers a runtime reallocation function, which can relocate a module/program with the help of information present in program's binary. This increases the

modularity and efficiency to detect bugs as all the modules including device-drivers are executed only after rooting the kernel, whereas with TinyOS, all are combined and it becomes hard to find a bug. However, this increases the execution time due to an overhead of loading and unloading modules.

Like in TinyOS, Contiki kernel is event-driven. A process scheduled to execute only if it triggers an event to either the event handler or polling handler. But the event handler and polling handler are not preempted and can make use of the same stack. But, unlike in TinyOS, the event handler cannot be preempted until and unless an event is sent to interrupt handler to cause the interrupt which can be prevented using an alternative way, which is to send the request to the kernel to start scheduling rather than interrupting current execution.

Furthermore, Contiki supports protothreads, which are lightweight stackless threads designed for severely memory constrained systems. A protothread consists of 2 main elements an event handler and a local continuation. An event handler is the one associated with the process, whereas the local continuation is used to save and restore the process state when a blocking API is invoked. The main benefit of protothreads is that it provides conditional blocking inside a process event-handler. As discussed above event handlers cannot block as they are non-preemptive and in case if an event-handler did not return, it would prevent the scheduling of other processes. Hence protothreads are required to allow the system to run other activities when the code is waiting for something to happen.

Contiki supports powerful, low power Internet communication stack that includes Internet standards IPv6 and IPv4 along with the recent low power wireless standards, such as 6LoWPAN, RPL, and CoAP. Contiki can run in different types of embedded networked drives, such as and other arm-based devices. In addition, it is possible to port Contiki to new hardware with a little effort.

Moreover, Contiki provides the Cooja Network Simulator to set up a large wireless network as it requires a lot of time and effort to debug and deploy devices. Cooja allows simulator large-scale networks comprising 10s of nodes running on fully emulated hardware devices.

Like in TinyOS, Contiki has no explicit power-saving functions, if there are no events to handle, the CPU is put into sleep mode.

However, TinyOS is considered to be better suited when resources are really scarce compared to Contiki. Contiki might be a better choice when modularity and flexibility are more important. Also, Contiki offers ContikiSec, which provides confidentiality, authentication, and integrity in communications under the Contiki operating system

RIOT

The kernel of RIOT was first developed by several European academic institutions in 1999, and later by various companies, individuals and universities. It is an open source embedded operating system, written in ANSI C.

RIOT implements a microkernel architecture, inherited from Fire Kernel. In addition to Fire Kernel's original features, powerful C++ libraries such as, the Wise Leaf algorithm framework and TCP/IP network stack are added into RIOT.

RIOT is designed in a modular way. It also tries to limit dependencies between the modules as small as possible. Another benefit of modularity is the effect of bugs is limited in the module itself, like that of in Contiki.

In addition, the configuration of the system can be customized to meet a particular specification, thereby minimizing the size of the kernel. For instance, a RIOT kernel can require as small as a few hundred bytes of RAM and program storage.

RIOT supports a tickles scheduler, which can easily to switch to an idle thread, when there are no pending tests. In addition, the idle thread contains a function to decide this looping duration of a device, or system. This decision is made, based on the peripheral devices in use, this gives the advantage that the system can stay in the sleep state as long as possible. Subsequently, energy consumption can be significantly minimized. Only interrupts wake the system up from the idle state.

Interrupt handling is an important part of RIOT. Because failure to handle interrupts well has a serious impact on the performance of the system. RIOT provides an API which is similar to common sensor net programming model. Also, RIOT supports multithreading with zero-latency interrupt handlers and minimum context-switching times.

Similar to Contiki, the network stack in RIOT is also modular. This gives the flexibility to exchange protocol layers at any hierarchy. In addition, an adaptation layer is provided that offers an IEEE 802.15.4 complete interface. Also, RIOT supports different network protocols for resource constraint systems Such as 6LoWPAN and RPL. In addition, IPv4, IPv6, UDP, and TCP are also fully supported.

Another advantage of Riot is related to portability in using advanced features of processors and it supports various types of hardware such as 16 bit MSP430 or 32 bit ARM server. Moreover, RIOT provides a developer-friendly API, which makes building and deploying applications on top of RIOT more feasible.

Hence RIOT serves as real-time OS for IOT platform and way better compared to TinyOS and Contiki. With Ubuntu Snappy which is introduced in next section, RIOT now offers UI interface as well.

Ubuntu Core

Ubuntu Core is a tiny, lightweight, transactional version of Ubuntu for IoT devices and large container deployments. It is designed to run securely on autonomous machines, devices, and other internet-connected digital things. Ubuntu Core Separation of OS and application files as a

set of distinct read-only images, to easily and securely add multiple apps and functionalities onto a single device.

Usage of IoT devices increased more and more with the Software Applications being deployed for each of this device, making it easier for the user to access and control devices remotely. Ubuntu offers App store for these so-called Software-defined devices. It provides Snaps, a new and simple application packaging system that Linux offers, to make it easier for developers to build and maintain apps within an application store model.

Snaps offer strict security, transactional updates, automatic rollbacks, save disk space using compression techniques and they are easy to create and distribute on any device. Snapcraft is the command line tool for writing and publishing your software as a snap. Snapcraft works with many languages such as python,Node.js, Go etc., to make packaging and installing software easy. Hence Snaps are widely adopted by industry leaders.

Ubuntu-core supports multiple-protocol for network stack such as Modbus, CAN, Zigbee, COAP or in the cloud MQTT, AMQP, making easier for developers to build apps for edge gateways. Ubuntu core offers edge gateways support where gateways connect sensors and actuators to clouds while providing edge analytics and intelligence.

It does background decision making to the gateway to ensure continuity and low-latency, and reduce energy-consumption for low-value data. Also with strict app confinement, multiple applications to run on the same hardware reducing the memory usage. Like for applications Ubuntu core offers a secure, reliable and remotely upgradeable platform for the intelligent edge as well.

Thus, Ubuntu core supports the wide range of applications such as Digital signage, Robotics, and Edge Gateways with the support of it's Cloud platform. Unlike above-mentioned TinyOS, Contiki and RIOT it is not restricted to the specific domain. It comes with a lot of flexibility and security to support real-time applications on IoT devices.

Android Things

Android things formerly known as Brillo is perfect for developers looking for the solution to build smart devices using Android APIs and Google services. It is Android based embedded operating system platform by Google announced in 2015.It is aimed to use in low power and memory constrained iot devices.

The best part is which are built from different platforms. Brillo brings the simplicity and speed of software development to hardware for IOT with an embedded OS, core services, developer kit and console.

One of the most interesting features of Android Things is WEAVE, which is a communications platform comprising both an SDK (software development kit) and cloud-based server that

allows device makers to connect their gadgets to Google's cloud services to enable features like voice recognition and data analysis.

Weave SDK will be embedded in the devices for local and remote communication. It is already used by device makers like Philips Hue and Samsung SmartThings and Google says others are implementing it including Belkin's WeMo, LiFX, Honeywell, Wink, TP-Link, and First Alert.

Changes in the Weave platform will make it easier for heterogeneous devices to connect to the cloud and interact with services like the Google Assistant. Google will also be adding more ready-made schemas to its Weave SDK to support a wider variety of IoT device types.

Also, Android Nest provides different IoT devices like Thermostats, Cameras, Doorbell and Alarm System that uses state-of-the art Weave Communication technology.

Weave security is independent of the underlying network. Every interaction between products, apps and cloud services is secure. Weave makes devices to connect easily, securely adds the new device to IoT network.

These devices, or *things*, connect to the network to provide information they gather from the environment through sensors, or to allow other systems to reach out. It becomes tough to handle such huge amounts of data coming from an ecosystem of devices. Here comes Google Cloud IOT core, which provides a fully managed service for managing devices. This includes registration, authentication, and authorization inside the Cloud Platform resource hierarchy as well as device metadata stored in the cloud, and the ability to send device configuration from the service to devices.

Android things also support tensorflow which enables it to deploy applications involving techniques like machine learning, computer vision on to devices in IoT Platform.

With these abilities Android Things now competes in the market as one of the top Operating System for IoT platform. However, there are still issues like Security updates and more yet to be improved by Google.

Conclusion

Designing an IoT device is very challenging in regards to choosing the suitable operating system, which must be capable to deal with many requirements of hardware and applications scenarios.It is difficult to answer which OS is most used for IoT.It would be misleading to assume one particular OS would serve all the IoT devices. There are million things that are not connected to internet and cannot make use of OS that has large RAM and power consumption.

Future Scope

Unlike personal PCs and mobile devices, wide variety of operating systems can be used in IoT market depending upon the need and application considering their pros and cons in the relevant scenario and uses. With increasing features in the Iot devices such as speech

recognition, natural language processing, and so on ,enabling smart homes to smart cities, will lead to increase in security vulnerabilities. The efficient solution to increase security would be designing a robust operating system.

Hence without proper operating system and security design, IoT can be dangerous sometimes when misused, so research on restricting the security measures and proper constraints, immediate need for encryption schemes and standardization is utmost important for future of IoT applications and efficient use of technology.

References

[1]Comparison of Operating Systems TinyOS and Contiki: Tobias Reusing
[2]TinyOS Programming,Philip Levis and David Gay
[3]https://devopedia.org/iot-operating
[4] OS for the IoT : Goals, Challenges, and Solutions: Emmanuel Baccelli Oliver Hahm Mesut Günes Matthias Wählisch Thomas C. Schmidt

[5]Operating Systems for Low-End Devices in the Internet of Things: a Survey: Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, and Nicolas Tsiftes