ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 5, Issue 5, May 2016

A Comprehensive Review on Embedded Hypervisors

Mounika M, Chinnaswamy C N

***Abstract*— Hypervisors and virtualization are common terms in the world of computing. Their references are generally associated with general purpose computers which have no resource constraints. However in an embedded environment with strict constraints on resources and computing power, virtualization becomes a challenging task. This paper gives an overview about the concept of virtualization and emulation as well as the background associated and the types of virtualiztion. It aims at discussing the concept of embedded hypervisors, comparing it with general purpose hypervisors and giving an account of its advantages, drawbacks and use cases.**

***Index Terms*—Embedded, emulation, hypervisor, virtualization, virtual machines.**

I. INTRODUCTION

Virtualization in the general terms refers to the act of creating or using a virtual version of a resource rather than the physical one. The resources that can be virtualized include computer hardware platforms, operating systems, storage devices or network resources. The definition of virtualization changes with the context and the type of resource being virtualized. One will often find references to terms like hardware virtualization, storage virtualization, server virtualization, memory virtualization or network virtualization. The definition of all of these terms is a context specific variation of the basic definition of virtualization.

A virtual machine (VM) is commonly defined as a software program that not only exhibits the behavior of a separate computer, but is also capable of performing tasks such as running applications and programs like a separate computer. However, this definition varies depending on the context. VM are also defined as the emulation of a particular computer system or architecture. By definition an emulator is hardware or software that enables one computer system to imitate another computer system. The emulated system is called the guest system and the system on which it is being emulated is called the host system. The concept of emulation allows software or a peripheral designed for the guest system to be executed on the host system. The emulator along with the lower level software and hardware are at times called a virtual machine.

Manuscript received May, 2016.

***Mounika M**, PG Student, Department of Information Science, The National Institute of Engineering Mysuru, India*

***Chinnaswamy C N**, Associate Professor, Department of information Science, The National Institute of Engineering, Mysuru, India.*

On an overview virtual machines are divided into two types: Process virtual machines [1] and System virtual machines [2]. A process VM is a virtual platform that executes an individual process. This type of VM exists solely to support the process; it is created when the process is created and terminates when the process terminates. In contrast, a system VM provides a complete, persistent system environment that supports an operating system along with its many user processes. It provides the guest operating system with access to virtual hardware resources, including networking, I/O, and perhaps a graphical user interface along with a processor and memory

Virtual Machine Monitors or hypervisors are a piece of software or firmware that helps create and run virtual machines. The hypervisor presents the guest system as an instance of the virtual machine. Hence, multiple instances of different guest systems can be run on the virtualized system.

A hypervisor is used when one needs to operate multiple systems on the same physical hardware. This partitioning is helping while performing system testing because new software or applications can be tested without affecting production. The process increases hardware efficiency and resource optimization through the operation of multiple workloads on each host.

This paper is a survey on hypervisors and embedded hypervisors in specific. It traces the evolution of virtualization as a concept in section II. Section III lists and describes the types of virtual machine monitors based on different parameters. The next section specifically discusses embedded hypervisors and section V lists the recent work done on embedded hypervisors. The next few sections give the use case of such systems, its advantages and drawbacks, followed by the conclusion in the last section.

II. BACKGROUND

Virtualization began in the 1960s, as a method of logically dividing the system resources provided by mainframe computers between different applications. As a technology, hypervisors were introduced nearly 50 years ago and are not really a new technology. The first recognizable products based on hypervisors had one main stand out motive for their development- making the best possible use of costly resources. As a result of using hypervisors the expensive hardware systems were more efficiently and economically used and software could be seamlessly executed on new hardware by emulating the old one.

Virtualization began and evolved on larger IBM mainframes, but the trend followed the evolution of

computing into servers, desktops, and recently embedded devices. In the 1990s, virtualization grew outside of the mainframe computers and servers. The figure below [3] is from a whitepaper on embedded virtualization from IBM that shows the timeline of evolution of the concept of virtualization.

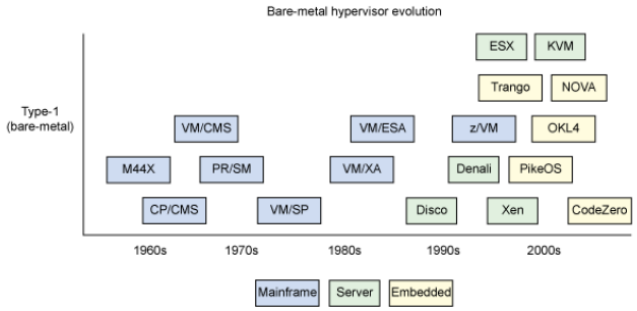

Fig 1: Brief timeline of type-1 hypervisors

III. Types Of Hypervisors

Hypervisors are generally classified into two types. This type of classification was introduced by Gerald J. Popek and Robert P. Goldberg in their 1974 article titled "Formal Requirements for Virtualizable Third Generation Architectures" [4].

A. Type-1 or Native Hypervisors

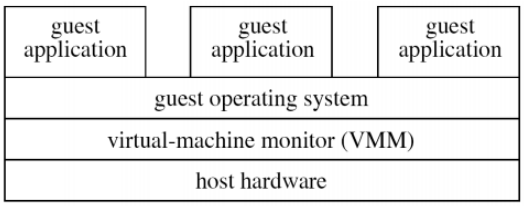

Fig 2: Type-1 Hypervisor

Type-1 hypervisors are those which run directly on the host system's hardware. They control and manipulate the hardware and also manage the guest systems. These types of hypervisors are also called bare-metal hypervisors or hardware virtualization engine. These virtual monitors are a slim layer of software which runs directly on the hardware and creates an environment in which the guest operating system can execute. They are generally preferred for embedded devices as they have minimum performance implications. The first hypervisors, which IBM developed in the 1960s, were native hypervisors.

B. Type-2 or hosted hypervisors

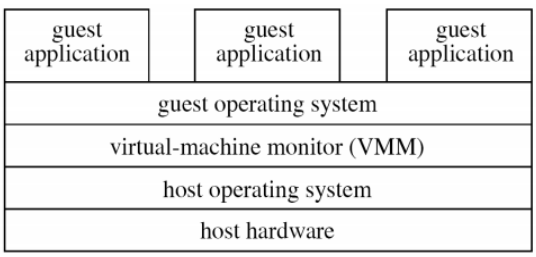

Fig 3: Type-2 Hypervisor

Type-2 hypervisors run on a conventional operating system that resides on the hardware. They function similar to other computer programs or applications running operating system. They are hence called hosted hypervisors. They are responsible for abstracting the guest operating system from the host operating system. Some famous examples of such hypervisors are VMware Workstation, VMware Player and VirtualBox. These types of hypervisors have performance implications as they are routed through the host operating system to gain access to the hardware.

A Type 1 hypervisor provides better performance and greater flexibility due to the less overhead involved in running the hypervisor itself. In contrast Type 2 hypervisors involve the hypervisor itself running on another operating system which increases the processing time and thereby leading to a higher overhead. Typically, a Type 1 hypervisor is more efficient than a Type 2 hypervisor, yet in many ways they both provide the same type of functionality because they both run the same kind of VMs.

IV. Embedded Hypervisors

The requirements to run virtual machines on desktop or enterprise operating systems are very different from the requirements on resource constrained microcontroller or in an embedded environment. An embedded hypervisor is designed to meet the needs and restrictions associated with embedded systems and real-time operating system environments [3]. The resource-constrained nature of many embedded systems, especially battery-powered mobile systems, imposes a further requirement for small memory-size and low overhead. The embedded world being extremely use case specific uses a wider variety of architectures and less standardized environments.

Other differences between virtualization in server/desktop and embedded environments include requirements for efficient sharing of resources, high-bandwidth, low-latency inter-VM communication, a global scheduling and power management.

Some parts in the literature refer to embedded hypervisors as Type 0 hypervisors or un-hosted hypervisors. The coining of these terms is related to the fact that embedded hypervisors are closer to the hardware when compared to Type 1 or Type 2 hypervisors. As shown in Fig 4, Type 1 hypervisors are often thought of to be integrated into the host operating system and in comparison embedded hypervisors run without a fully fledged operating system on the native device which means that it does not need an operating system to be integrated into and hence the tem un-hosted.

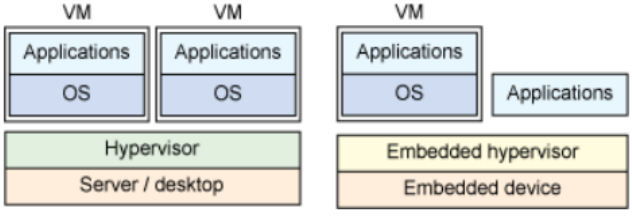

Fig 4: Bare-metal operating systems and hypervisors [3]

Although the features of embedded hypervisors vary due to the context and device specific nature, some of the common features are native support for different processors, default system driver compatibility, direct access to system resources and strong protection against viruses and malware because they are directly embedded into the hardware.

A. Requirements and Motivation

The porting of hypervisors on embedded environments comes with many restrictions. While efficiency is the main objective embedded hypervisors deal with added constraints outside of traditional virtualization environments. Memory sharing and management is extremely critical as it is the most severe constraints on embedded devices. The limited memory raises a need for extremely small and efficient hypervisors. The sleek design makes it easier to validate the code and prove that it is bug free making the hypervisor more secure.

The smaller the hypervisor, the more secure and reliable the platform can be. This is because the hypervisor is typically the only portion of the system to run in a privileged mode instruction, which serves as what is known as the trusted computing base (TCB) and leads to a more secure platform.

One of the ways to ensure small and efficient hypervisors is by using microkernel and separation kernel based designs. Therefore embedded hypervisors usually have a small memory footprint from several tens to several hundred kilobytes, depending on the efficiency of the implementation and the level of functionality provided. An implementation requiring several megabytes of memory (or more) is generally not acceptable.

An embedded hypervisor needs to be in complete control of system resources, including memory accesses, to ensure that software cannot break out of the VM. A hypervisor therefore requires the target CPU to provide memory management support (typically using an MMU).

Embedded hypervisors that are built for sharing a hardware platform with multiple guests and applications and also allowing them to interact are referred to as Embedded Communication Hypervisors. This channel for communication is both efficient and secure, permitting privileged and non-privileged applications to coexist. This also helps in providing license segregation. It permits proprietary software and open source software to coexist in isolated environments.

Finally, the embedded hypervisor must support scheduling with real-time capabilities. In the case of handsets, the hypervisor can share the platform with core communication capabilities and third-party applications. Scheduling with real-time characteristics allows the critical functions to coexist with applications that operate on a best-effort basis.

In most cases an embedded hypervisor is considered a type-1 hypervisor but a strict differentiation as to which type of hypervisors the belong to is debatable. With the before mentioned background one can conclude that some of the basic requirements of embedded hypervisors are as follows:

- Small size and sleek implementation.
- Support for independent execution but interactive coexistence of applications in a secure environment.
- Low latency communication and switching between system components.
- Being able to meet real time requirements of embedded applications and minimum affect on the performance.
- Minimum affect on system resources and fast response time.

B. Implementation

An embedded hypervisor typically deals with one or more virtual machines meant for embedded devices and ensure they interact smoothly. This process may or may not involve emulating the native hardware.

When a VM provides a virtual platform, guest software has to be ported to run in this environment. This virtual platform is independent of the underlying hardware and is capable of mapping the virtual system to any platform supported by the hypervisor.

Embedded hypervisors don't always work on full virtualization schemes and employ either para-virtualization where the hardware does not assist a particular feature, and involves often extensive modifications to core architecture in order to support core of guest kernels.

Emulation of hardware at the register level is rarely seen in embedded hypervisors as this is very complex and slow. The custom nature of embedded systems means that the need to support unmodified binary-only guest software which require these techniques is rare.

Implementations for embedded systems applications have most commonly been based on small microkernel and separation kernel designs, with virtualization built-in as an integral capability.

One of the most important functions required in an embedded hypervisor is a secure message-passing mechanism, which is needed to support real-time communication between processes. Inter-process communication can be used for this purpose but the IPC mechanism has to be highly optimized for the restraints in the embedded environment to ensure minimum latency and low impact on the system performance.

V. Related Work

Virtualization was introduced in the late 1960s or early 70s as a quick hack to counter the ridiculously high prized hardware. With the silicon revolution and as the cost of computers and electronics reduced, virtualization was used to only provide compatibility between devices.

Over the last couple of years the concept of virtualization is regaining its importance; however this time the reason is secure execution and partitioning. With microcontrollers getting more and more complex and seamless connectivity becoming an agenda high on the priority list, the concept of virtualization got a new and more customized meaning.

The work by the authors in [5] is one of the first on embedded hypervisors for consumer electronics. The paper examines a number of typical virtualization use cases from the consumer electronics domain, and the resulting requirements imposed on the hypervisor. The authors have presented the OKL4 hypervisor, a member of the L4 microkernel family, designed for embedded-systems use. It provides special emphasis on its security mechanisms, and its performance has been compared to a version of Xen.

In [6] the author, Rossier Daniel provides with a variant for the famous Xen virtualization for embedded ARM based applications. The paper presents a novel approach in virtualization mechanisms to address particular needs in embedded systems, such as dealing with heterogeneous ARM cores and keeping execution overhead as low as possible. The authors introduce EmbeddedXEN, a revisited virtualization framework based on the existing XEN hypervisor.

In [7] the authors of the paper look into virtualization for automobiles systems and use virtualization as a solution for the safety, portability, and maintainability and efficiency components of the non functional requirements of the automotive systems. Virtualization helps merge multiple safety-relevant sub-systems onto a single hardware platform and to implement strong separation. The paper describes and evaluates a microkernel approach to isolate safety-relevant automotive software virtual machines by using a Memory Management Unit less embedded hypervisor.

The work done by Reinhardt Dominik and Gary Morgan in [7] has been used as a starting point for several papers which discuss virtualization in an automotive environment. [8] Discusses virtualization as a method to isolate functions on a multi-core platform in an automotive embedded environment. It specifically discusses the sharing of IO resources for network peripherals and the overhead associated with these processes. The authors compare two approaches to share the virtual resources, first is software-based para-virtualization and second I/O virtualization built into the CAN controller itself.

Besides the direct relationship with cost reduction and improved resource utilization, virtualization enables the integration of real-time and general-purpose operating systems and applications on the same hardware platform. The resulting system may inherit deterministic timing characteristics for real-time along with a large software code base for general-purpose operating systems. However, the hypervisor must be carefully designed to take advantage of both types of operating systems. Moratelli, Carlos, and Fabiano Hessel in [9] propose an interrupt policy for an embedded hypervisor using hardware-assisted virtualization.

Up to now, automotive systems are not developed to run within virtualized environments. To satisfy the demands rising due to intelligent embedded systems, the quantity of electronic control units in cars has increased dramatically. OEMs tackle that trend by consolidating software on powerful multicore hardware platforms. However, current software solutions are mostly static and designed to run on limited platforms. As promising operating system for automotive, Linux comes into consideration, which seems to scale better than already existing solutions. [10] suggests embedded hypervisors as a solution to ease the migration process of older software parts and guarantee freedom from interference according to industry standards. It presents an approach to map communication channels of virtual automotive ECUs and connect them with their already existing CAN interfaces. For our analysis, we use the Xen hypervisor.

VI. Use Cases

Embedded hypervisors find applications in many fields most prominent of them being the automotive, industrial and medical fields.

The automotive field use multiple electronic systems each using controllers from different vendors or there is also a possibility for infotainment software, instrument cluster control and telemetric to all run on a single multi-core chip. Such a scenario raises the need for multiple operating systems to run on the chip, for example an RTOS for instrumentation and GPS and Linux for audio. The use of hypervisors makes sense in this case.

Similarly, for industrial applications there is a requirement for an RTOS For industrial applications (factories, mines, power plants etc.) there is commonly a need for real time control (RTOS) and sophisticated networking (Linux). In addition, in recent years there has been an increasing concern about cyber-attacks on or other introduction of malware into control systems. A hypervisor is the ideal way to separate systems and maintain security.

Medical systems have a mixture of real-time (patient monitoring) and non-real-time (data storage, networking and user interface) functionality, a hypervisor can provide both the functionalities. It also ensures secure and independent execution of the real-time and non real-time applications

The examples given above are one of the many uses cases of embedded hypervisors. The main reason for using embedded hypervisors is the sandbox environment that provides a secure environment for the application to be executed. Apart from that they can prove extremely efficient in providing platform independency and backward compatibility.

VII. Advantages And Drawbacks

One of the biggest advantages one gains by using a hypervisor for embedded devices is security. The use of a hypervisor provides a sandbox environment which helps providing application and process independence. It can make migration between processors of different instruction set architectures possible without overhead (if the right precautions are taken).The process of emulation provides platform independency and the ability to run the new version of programs on legacy systems thereby providing backward compatibility.

One of the major issues with virtualization or using virtual machine monitors is the overhead involved in performing the virtualization procedure. The process of virtualization might involve emulation which can be time consuming. Another issue which comes along with this topic is the performance snag. Depending on the way instructions are handled by the hypervisor and are translated by the emulators, the performance of the embedded device may take a massive hit.

Although there are multiple ways using which the performance can be enhanced it, virtual machines will always be slower than the native machine.

On comparing the pros and cons of virtualization for embedded devices one can conclude that the process of virtualization can be used to provide platform independency and a secure environment for execution but this comes at a cost. Hence, one must ensure that embedded hypervisors are used only when absolutely needed and the overhead and performance dip is well compensated by the use of caching or multi-core environments.

VIII. CONCLUSION

The design of hypervisors for desktops and serves differs from the design used for embedded devices. The resource constraints raise a requirement for sleek and efficient design, keeping the code minimalistic and more secure. The use cases for embedded hypervisors are on the rise, with internet of things and seamless connectivity becoming an integral part of the technology revolution embedded virtualization has come into the picture as cost effective solution for connectivity problems. Emulation done as a part of building hypervisors for embedded devices comes in handy as it leads to platform independency as a side effect. Microcontroller vendors have introduced hypervisors with different designs in accordance to their processor architectures. Despite of the availability of many proprietary and open source hypervisors, embedded hypervisors are still a topic for research due to their context specific nature.

ACKNOWLEDGMENT

We are greatly thankful to our family and friends for the continuous support that has provided a healthy environment to drive us to do this project. We also thank the Principal and the Management of NIE for extending support to this work.

REFERENCES

[1] Overby, Eric. "Process virtualization theory and the impact of information technology." Organization science 19.2 (2008): 277-291.
[2] Smith, James E., and Ravi Nair. "The architecture of virtual machines."Computer 38.5 (2005): 32-38.
[3] Jones, M. Tim. "„Virtualization for embedded systems." The how and why of small-device hypervisors", IBM developerWorks (2011).
[4] Popek, Gerald J., and Robert P. Goldberg. "Formal requirements for virtualizable third generation architectures." Communications of the ACM 17.7 (1974): 412-421.
[5] Heiser, Gernot. "Hypervisors for consumer electronics." Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE. IEEE, 2009.
[6] Rossier, Daniel. "EmbeddedXEN: A Revisited Architecture of the XEN hypervisor to support ARM-based embedded virtualization." White paper, Switzerland (2012).
[7] Reinhardt, Dominik, and Gary Morgan. "An embedded hypervisor for safety-relevant automotive E/E-systems." Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on. IEEE, 2014.
[8] Herber, Christian, et al. "HW/SW trade-offs in I/O virtualization for Controller Area Network." Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE. IEEE, 2015.
[9] Moratelli, Carlos, and Fabiano Hessel. "Hardware-assisted interrupt delivery optimization for virtualized embedded platforms." 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS). IEEE, 2015.
[10] Reinhardt, Dominik, et al. "Mapping CAN-to-ethernet communication channels within virtualized embedded environments." Industrial Embedded Systems (SIES), 2015 10th IEEE International Symposium on. IEEE, 2015.

Mounika M is a PG student in the Department of Information Sciences at The National Institute of Engineering, Mysuru. She has a Bachelor's degree in Electronics and Communication.

Chinnaswamy C N is an Associate Professor in the Department of Information Science & Engineering. He has received his M.Tech from VTU and B.E. from University of Mysore. He is pursuing his Ph.D and his teaching and research interests are in the field of Computer Networks and Internet of Things.