REAL TIME HYPERVISOR AND VIRTUALIZATION – A SURVEY

Rushang Vinod Vandana Karia
Arizona State University
Rushang.Karia@asu.edu

Abstract
In this paper we emphasize the concepts of hypervisor when applied to real-time systems. Real-time environments are wide and varied in requirements. They often have more strict and stringent requirements that must be imposed at all times. One of the most important features is predictability. Therefore the hardest part of such systems is verification to ensure the satisfaction of this property at all times. Hypervisors often introduce overhead into the system to provide more generality and host various different types of applications. As such, their adherence to the requirements of real time systems can be challenging.

Keywords
Real-time, hypervisor, virtualization, virtual machine, virtual machine monitor, real-time issues, Embedded Systems

Introduction

The dictionary definition of virtualization is *"something which is not independent but which both works and appears to the user as a single logical entity"*. A hypervisor is an entity which is responsible for maintaining the property of virtualization. In the context of operating systems, the most direct example can be virtual memory. In this case the memory hierarchy is modelled as a virtual machine to the operating system and the hypervisor is the memory management unit of the underlying architecture.

Virtual machines and hypervisors have been existent for quite some time now. Many of them have brought commercial success to their creator. VMWare and Oracle are among the major companies who develop and market hypervisors with their products of vSphere and Virtual Box respectively. While these products virtualize Operating Systems, the concept of virtualization is more general. For example, a server could be virtualized or a network. Virtualization can be used to provide a more abstract view of the underlying model.

A hypervisor is the "manager" of virtual machines. It controls the operation of the virtual machines and resolves any errors or conflicts that are generated.

The rest of the paper is organized as follows. In section I we elaborate further on a hypervisor and virtual machine specifically those applicable Operating Systems. In section II we discuss the issues in implantations of hypervisors. In section III we provide a summary of real time systems and discuss the implementation and issues involvement in the development of hypervisors for real time environments. In section IV we do case studies on various real time hypervisors. In section V we present ways to evaluate the performance of real time hypervisors and discuss some benchmarking performed on RT-Xen.

I
BACKGROUND

1. Virtual Machines (VM's)

Goldberg [1] describes a virtual machine as "*A system which is a hardware-software duplicate of a real existing machine, in which a non-trivial subset of the virtual machine's instructions execute directly on the host machine"*. This virtual machine operates at its level of abstraction. It has no knowledge of the lower layers. So an Intel virtual machine would assume that the memory is arranged in little-endian format. It could be the case that the memory is actually big-endian. Virtual Machines are what the users see and use.

There exist two types of Virtual Machines. Each type has its own uses, advantages and disadvantages. The architecture of both the machines are quite due to the different requirements imposed by each one.

1.1. Types of Virtual Machines

A. System Virtual Machines

These are machines which provide a complete platform-wide implementation. They are usually used to provide a virtual environment to program when the hardware is not present or currently in development. They can mean complete Operating Systems which run on the underlying hardware. We will restrict the term virtual machines to mean system virtual machines unless stated otherwise.

B. Process Virtual Machines

These are machines which run as applications and support the execution of a single program. The best well-known example is the JAVA Virtual Machine (JVM). They are processes of the target OS and provide an environment different than that of the OS to the program.

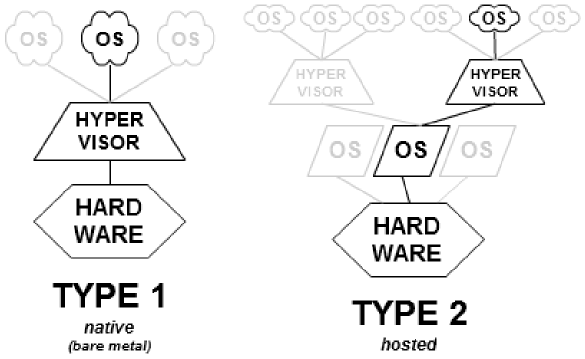
1.2. Emulation and VM's

Besides the many similarities between emulation and VM's there exists one key difference separating the two. VM's can run directly on the physical hardware thus avoiding the emulation tax-the translation of the emulator instructions to the physical machine instructions. However it must be kept in mind, that even though they can run on the physical hardware, they are still scheduled by an upper-level construct. As such they have a faster speed at the cost of versatility.

1.2. Hypervisor

Also known as a Virtual Machine Monitor (VMM) a hypervisor is a program that provides and regulates access to the virtual machines. Since in its context virtual machines are processes to be scheduled, all problems arising in multi-programming environments are handled by it. Further each process can be an Operating System (OS) that directly talks to the hardware. So the hypervisor has to deal with all interrupts, network packets and pass them on to the appropriate virtual machine.

Types of Hypervisors



A. Bare-Metal Hypervisor (Type 1)

These run atop the hardware directly and control access the VM's. The hypervisor has a compatibility package in which it translates instructions for different architectures into instructions for the underlying architecture.

B. Hosted Hypervisor (Type 2)

These kind of hypervisors run as processes of Operating Systems. They translate requests for memory and resources into system calls for the host OS kernel.

C. Type Zero Hypervisor

The hypervisors discussed earlier are stand-alone and do not receive any help from the hardware. Nowadays, leading processor companies are providing built in support for virtualization. This reduces security loopholes as well as the size of the hypervisor. Also the hypervisor may run in the same mode as the kernel or the processor may provide additional features to the hypervisor. The reader is encouraged to view [3, 4] for a more detailed treatment of this type.

II
ISSUES IN IMPLEMENTATION

2. Implementation of VMM's

Hypervisors need to ensure that the stability of the underlying system is not affected by their presence. [7] prove 3 properties that every hypervisor must enforce to ensure correct operation and efficient implementations. There are many concerns in the implementation of a VMM. The first is performance. While some overhead is unavoidable the VMM must be able to efficiently allocate resources to ensure maximum efficiency of the VM's and also of the host OS if it is a hosted hypervisor. This is further complicated by the underlying hardware architecture. As an example, the implementation of a hypervisor on an x86 architecture since the x86 does not trap on some instructions such as register reads. [9] discuss important benchmarks and compare when it is beneficial to implement to a VMM.

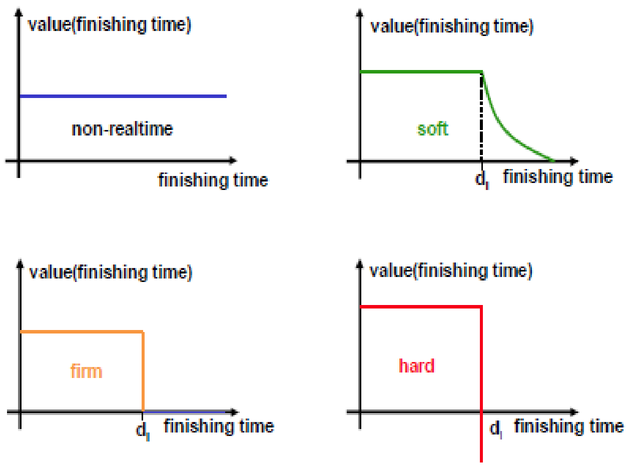
Perhaps one of the other most important concerns other than performance is security. There has been a major debate whether the security issue of a hypervisor is of a major threat. Some researches argue that since the hypervisor can run at a level lower than the Operating System, it is incapable of detecting it. As such malware could easily install itself below the Operating System and the OS would have no way of detecting it. Others argue that it might be possible to detect such a rootkit. Researchers from Microsoft and North Carolina State University have successfully demonstrated the detection of malware below the OS level. [10] lists out threats detected and possibly fixed in various VMM's and [11 ,12] discuss a possible implementation of security in a hypervisor.

III
REAL TIME HYPERVISOR

3. Real Time Operating Systems

A Real Time Operating Systems (RTOS) is an operating system with strict timing requirements. It has to be able to guarantee timely execution of tasks. This imposes constraints on the duration of the critical sections, context switches and resources. Often RTOS are used in the control of various embedded systems. Time critical applications like a patient-monitoring system often use an RTOS. An RTOS can be divided into two classes. A soft RTOS is one which can occasionally miss deadlines. The system performance may degrade gradually but it might not fall abruptly. On the contrary a hard RTOS is one who must guarantee that all deadlines are met. Any failure of meeting any one of the deadlines can lead to catastrophic failures of the system.

The following graph highlights the difference between the value of a missed



deadline in hard real time systems and soft real time systems.

3.1. Scheduling Algorithms

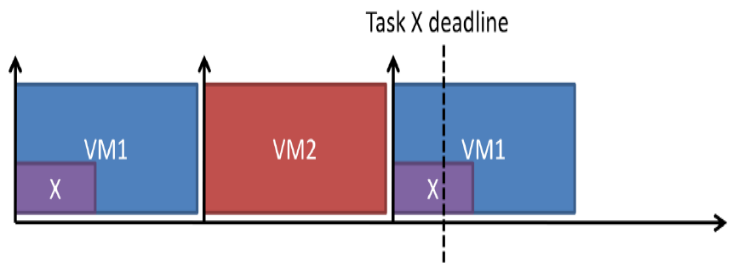
The heart of an RTOS is the scheduling algorithm. Since the RTOS must guarantee the timely execution of tasks, verification of an RTOS can be only shown by the verification of the scheduling mechanism that it uses. A number of scheduling algorithms have been proposed in the literature. Particularly of importance are optimal scheduling algorithms. A scheduling algorithm is deemed optimal if it can successfully schedule a set of tasks without missing their deadlines provided that such a schedule exists. While the problem is easy to state it is not so easy to verify since a varied number of factors often come into the middle. Earliest Deadline Scheduling (EDF) and Rate Monotonic (RM) algorithms are two popular algorithms. Both are optimal under certain conditions. Refer to [13] for a textbook treatment of various scheduling algorithms and their verification strategies.

Another important issue that arrives in the implementation is memory allocation and locking. A RTOS is expected to have a very high MTBF (Mean time between failures). This is the outcome of very high availability. Since an RTOS is often used in an embedded application, it is important to conserve memory. Thus memory leaks must be avoided in an RTOS. Also it is important to note that the stability of an RTOS must be high. Therefore the issue of dangling pointers must also be handled. Since most scheduling algorithms can only be verified if some parameters like the WCET (Worst case execution time) is known, it is important to ensure that the blocking time of a process is predictable. Thus all high priority tasks and interrupts must be able to guarantee a certain upper bound on locking and other asynchronous events like network latency.

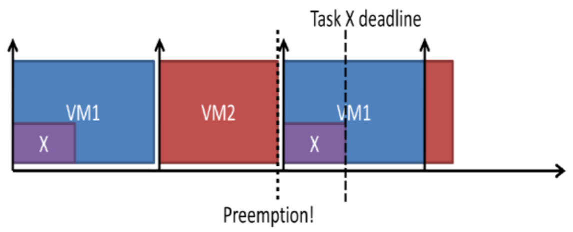
These factors make the implementation of an RTOS difficult, since we want the RTOS to be flexible. A notable problem [14] was priority inversion that affected the NASA Mars Pathfinder. While the mission was not affected since a fix was available, this incident was an eye-opener into the problems of Real Time Systems. But these parameters often lead to the design of very conservative systems.

3.2. Real Time Hypervisor

Real time hypervisors are VMM's created for Real Time Systems. Due to the various limitations imposed by real time systems like a very small footprint, predictability requirements, they must be designed taking into account all these limitations.



The issue of scheduling becomes evident in a real time hypervisor. Since a VM is generally construed as a black box by the hypervisor, with the VM responsible for scheduling of its own processes. As such it is not clear about the implementation of the scheduling



mechanism. In the above figure, Task X of VM1 fails to meet its deadline sine the hypervisor is not aware of the individual processes within a VM.

The above figure shows a correct implementation of the hypervisor in which it is aware of the deadlines of each individual tasks within the VM's. This is difficult to implement and the complexity rises with every addition of a VM. Thus the amount of bookkeeping is bound to increase unless a more novel approach is discovered.

Another difficulty is energy consumption, a translation of instructions normally is inefficient, furthermore the difficulty of porting different applications of different architectures provides significant overhead if they are not compatible. Also Inter-process communication costs must be made irrelevant as discussed in [15]. Another requirement is efficient resource sharing. A large network latency does not fit the real time model but it is needed in most real time applications. The Virtualization concept has not been able to effectively handle this type of situation. The Embedded Microprocessor [16] consortium is responsible to benchmark the performance of a hypervisor in a real time system.

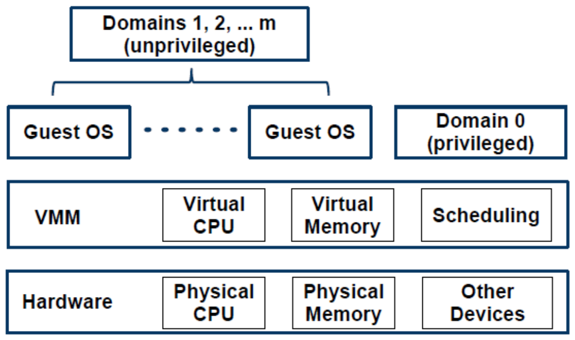
3.3. Para-virtualization

The concepts stated uptil now assumed that the VM behave as they are running on the hardware for which they were developed. This is called full virtualization. However this method is generally slower and complex to implement. Researches have come up with the concept of para-virtualization in which the guest OS knows that it is not running on the hardware that it is supposed to run on. In full-virtualization any request for any hardware resource would be directly performed by the VM which would cause a trap and the host would then perform the operation. In a para-virtualized scenario, the VM asks the hypervisor to perform the operation for it.

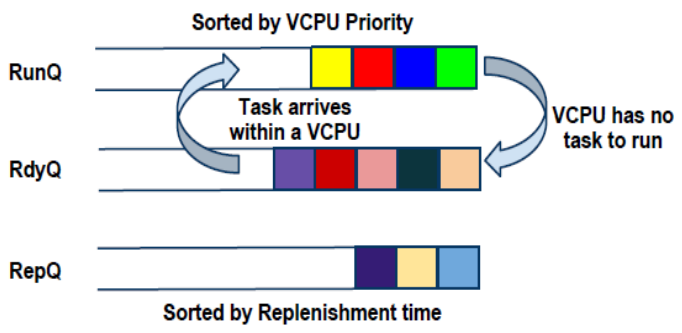
IV
CASE STUDIES

4.1. RT-XEN

Xen [17] is the most widely used open-source hypervisor. It is a bare metal hypervisor and it controls resources using a technique called para-virtualization wherein a single instance is created at boot and this instance is responsible for the spawning of the other VM's. The special domain usually called



domain 0 runs in privileged mode of the architecture and therefore has access to all the resources. Here, Domain 0 runs in privileged and it is responsible for controlling the behavior of the other domains. Xen introduces the concept of Virtual CPU (VCPU) so that quanta can be effectively distributed among the different VM's. A VCPU is just like a process in contemporary operating systems. To support real-time operating the current scheduling mechanism of VCPU is expanded to contain queues representing the state of each VM.

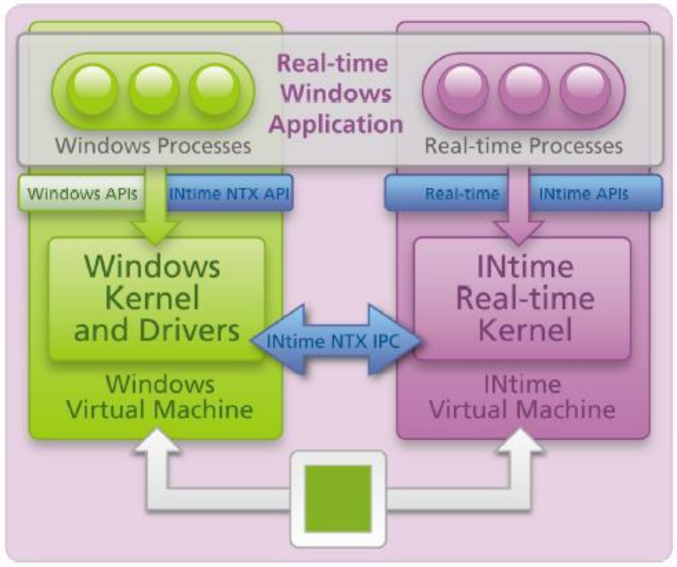


Each physical core gets a set of 3 queues. The operations of the RunQ and ReadyQ are obvious and are the same as in existing operating sytems. The RepQ (Replenishment Queue) is used to store replenishment information for the VM. At each scheduling quantum the scheduler checks this queue, if any value has a higher priority than any of the other queues it is scheduled immediately. To enable flexibility, provisions for different kind of servers and scheduling mechanisms have been provided. RT-Xen supports Deferrable Server, Sporadic Server, Periodic Server and Polling server. The memory allocation, resource control are performed exactly the same way as in Xen.

[18] discuss some performance benchmarks on RT-Xen. They ran performance metrics for soft real time, hard real time as well overloaded conditions. Each of the server were tested under identical conditions. They show that different servers respond differently to different kind of VM's. RT-Xen can be perceived as a bridge between real-time scheduling and virtualization.

4.2. TenASys Real-Time Hypervisor

The TenASys is an x86 hypervisor capable of running both an RTOS and another OS at the same time. TenASys



differs in the method of allocating physical resources. The developers argue that the multiplexing of I/O in traditional hypervisors often makes the hypervisor indeterministic in behavior. They assert that since the multiplexed I/O implies a wait on all VM's trying to access the device, this wait can be as large as possible. They attempt to reduce this non-determinism by identifying the devices which can be multiplexed and those which should be exclusive to a single VM. Thus one can separate time critical resources and provide special handling for them. TenASys also locks a VM in RAM, in that a VM is never swapped to disk in favor of more frequent use of another VM. This ensures that every real-time event is serviced and deterministic timing requirements are met.

Their approach of isolation and special handling of time-critical resources is applied to frame-buffers, network cards and other resources. They state that by doing so they can guarantee optimum performance of the VM. The hypervisor boasts support for more than one RTOS to run within it.

Their approach to multi-core architectures is to run each OS on a separate core. This minimizes resource contention and simplifies the implementation of the hypervisor.

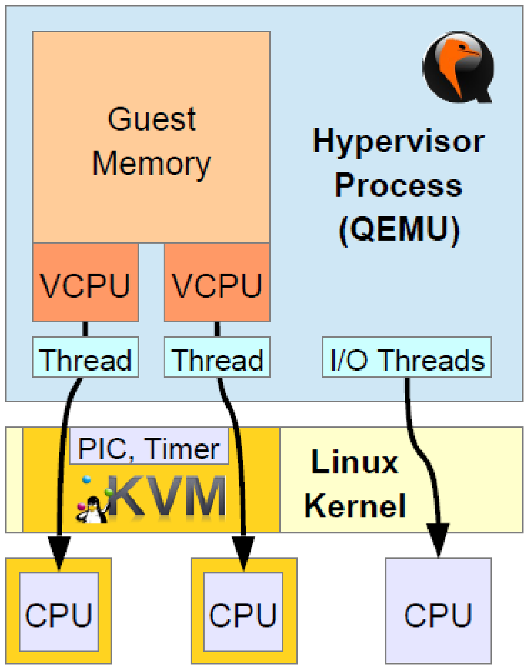
4.3. Linux as an RT-hypervisor

There has been considerable success in supporting real-time tasks in the Linux OS. Linux also supports many different architectures. The PREEMPT-RT patch [22] enables Linux to more demanding and stringent real time requirements. Using Linux as a hypervisor will require a kernel driver and user-space application. Linux supports KVM (Kernel based virtual machine).

The Linux KVM is a type 2 hypervisor running atop Linux. This simplifies the design of the KVM and enables it to focus on more important issues. Since it is implemented as a device driver, any process can talk to it using the same interface as those for char devices combined with IOCTL'S. This is normally combined to run with QEMU in the user-space. QEMU is a hardware emulator. Since QEMU does not include support for multi-threading the VM's are protected by a global lock thus limiting the device access capabilities of the VM's. This means that even if there are multiple instances of a device, only one device can be accepting it.

The performance of QEMU can be increased by setting it as the maximum priority task. However this includes the possibility of livelocks with the RT patch.

An example of a livelock possible with QEMU is provided in [20]. They describe a scenario in which the timer spins endlessly preventing any timer



interrupts to be issued. While the existence of a single big lock can only be mitigated by the inclusion of multi-threading support into QEMU, the KVM approach can provide for efficient implementations but its applicability to safety-critical systems is limited. For simple applications this approach looks the best.

V
TESTING OF A HYPERVISOR

Unlike General Purpose Operating Systems (GPOS), the testing of virtualized machines and VMM's is not so straightforward. While traditional testing techniques may be used, they might not be able to capture fully the performance of the virtualization layer.

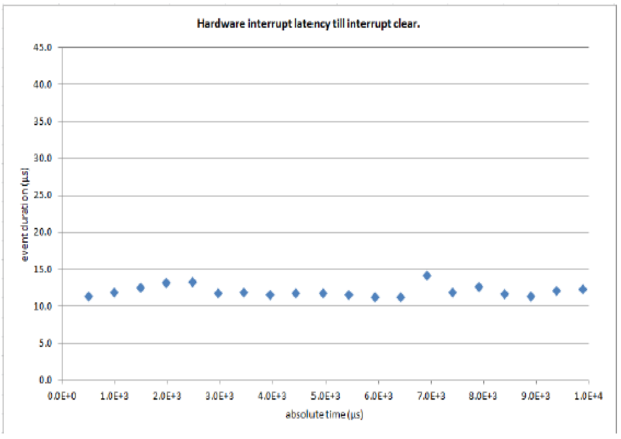
A number of factors play importance in the testing of a hypervisor. An excellent source for learning more about this subject is [24]. We discuss some points in [29] and replicate some of their results. They did some benchmarking on RT-Xen. Another study between Hyper-V and vSphere can be found at [25, 26].

5.1. VMmark

We first discuss a general framework for Real-Time testing, VMmark [28] is an industry standard benchmark methodology formed by the collaboration of leading providers of virtualization. The benchmark requires tests against popular workloads such as JAVA, Mail, File and Database server workloads. Existing benchmarks for these servers are generally modified to suit the VM profile. This reduces the development time for testing. Each VM is submitted a workload and the results are aggregated in the end. They are normally compared against standardized results.

5.2. RT-Xen Performance

In this section, we evaluate the performance of RT-Xen as done in [29]. The test cases were 2 VM's, domain 0 and another VM that were each designated a CPU. Dedicating an entire VM in such a way implies that it should not be preempted at any cost. This is accomplished by providing it a budget that is greater than its period.



Above are results that illustrate the time to process an interrupt in RT-Xen under the stated conditions. [29] conclude that this is correct behavior for a real-time hypervisor.

Another test that they ran was the processing of a clock-tick. Since this is critical to the functioning of an RTOS, they stated that RT-Xen does poorly when the budget is less than the period. This problem will get further aggravated if multiple VM's were to be scheduled on the same physical core. They conclude with further results and state that RT-Xen is a comparatively good VMM for real-time systems if certain parameters are considered.

Conclusion

Virtualization technology is a promising approach to large scale applications and servers. Most of the hypervisors we studied differ in their implementation of certain modules only which are modified to meet real-time requirements. A more novel approach is required to face the practical difficulties in implementation of real time VMM's particularly in embedded applications which are seeming to grow at an alarming pace. While there is much research to be done in virtualization, the technology developed for real-time systems can be used to enhance the performance of native systems even further.

References

[1] Goldberg, R. P. Virtual machines - semantics and examples. IEEE Comput. Soc. Conf., Boston, Mass. (September, 1971), 141-142.

[2] A Survey on Virtualization Technologies - Susanta Nanda, Tzi-cker Chiueh

[3] embedded.communities.intel.com/docs/DOC-7334

[4] http://embeddedinnovator.com/2012/06/the-rise-of-the-type-zero-hypervisor/

[5] http://en.wikipedia.org/wiki/Hypervisor

[6] Efrem G. Mallach - On the relationship between virtual machines and emulators

[7] Popek, G. J.; Goldberg, R. P. (July 1974). "Formal requirements for virtualizable third generation architectures". Communications of the ACM 17 (7): 412–421.

[8] Intel, “Intel Virtualization Technology," Intel Technology Journal, Volume 10, Issue 3, August 2006.

[8] Implementation of a Purely Hardware-assisted VMM for x86 Architecture - Saidalavi Kalady , Dileep P G, Krishanu Sikdar, Sreejith B S, Vinaya Surya, Ezudheen P. Y

[9] When Virtual Is Better Than Real - Peter M. Chen and Brian D. Noble

[10] web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-4993

[11] Architectural Support for Hypervisor-Secure Virtualization - Jakub Szefer, Ruby B. Lee

[12] Hypervisor-based Fault-tolerance – Bressoud T.C, Schneider F.B

[13] Real Time Systems – Jane S. Liu

[14] research.microsoft.com /en- us/ um/ people/ mbj/mars_pathfinder/mars_pathfinder.html

[15] Gernot Heiser - The Role of Virtualization in Embedded Systems

[16] http://www.eembc.org/benchmark/hyper_sl.php

[17] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield – Xen and the art of visualization

[18] Sisu Xi, Justin Wilson, Chenyang Lu, and Christopher Gill - RT-Xen: Towards Real-time Hypervisor Scheduling in Xen

[19] http://www.tenasys.com

[20] Jan Kiszka - Towards Linux as a Real-Time Hypervisor

[21] Internals of the RT Patch, Steven Rostedt, Darren Hart, Proceedings of the Linux Symposium, 2007.

[22] Real-Time Linux Wiki, 2009. http://rt.wiki.kernel.org

[23] events.linuxfoundation.org /sites /events /files /lcjp13_kiszka.pdf

[24] www.cc.iitd.ernet.in /misc /cloud / hypervisor_performance.pdf

[25]blogs.technet.com/b/keithmayer/archive/2013/09/24/vmware-or-microsoft-comparing-vsphere-5-5-and-windows-server-2012-r2-at-a-glance.aspx

[26]http://www.vmware.com/files/include/microsite/sddc/principled_technologies_vmware_vs_microsoft_tco.pdf

[27] http://www.vmware.com/pdf/vmmark_intro.pdf

[28] www.idt.mdh.se/ utbildning / exjobb /files / TR1180.pdf

[29] Hasan Fayyad-Kazan, Luc Perneel, Martin Timmerman – Evaluating the performance and behavior of RT-Xen