2010 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
VEHICLE ELECTRONICS & ARCHITECTURE (VEA) MINI-SYMPOSIUM
AUGUST 17-19, 2010 DEARBORN, MICHIGAN

VIRTUALIZATION – THE POWER AND LIMITATIONS FOR MILITARY EMBEDDED SYSTEMS – A STRUCTURED DECISION APPROACH

Mike Korzenowski
Vehicle Infrastructure Software
General Dynamics Land Systems
Sterling Heights, MI

ABSTRACT

Virtualization is becoming an important technology for military embedded systems. The advantages to using virtualization start with its ability to facilitate porting to new hardware designs or integrating new software and applications onto existing platforms. Virtualization is a tool to reuse existing legacy software on new hardware and to combine new features alongside existing proven software. For embedded systems, especially critical components of military systems, virtualization techniques must have the ability to meet performance requirements when running application software in a virtual environment. Together, these needs define the key factors driving the development of hypervisor products for the embedded market: a desire to support and preserve legacy code, software that has been field-proven and tested over years of use; and a need to ensure that real-time performance is not compromised.

Embedded-systems developers need to understand the power and limitations of virtualization. This paper presents virtualization technologies and its application to embedded systems. With the dominant market of multi-core processing systems, the need for performing specific hardware/software configuration and usages with relations to Platform Virtualization is becoming more and more prevalent. This paper will discuss different architectures, with security being emphasized to overcome challenges, through the use of a structures decision matrix. This matrix will cover the best suited technology to perform a specific function or use-case for a particular architecture chosen. The topics of the "Best Suited Technologies" to utilize when considering Virtualization will cover the following architectures: Virtualization – Hypervisors, Hyper-Threading, Single-Core CPU Architectures, Multi-core CPU Architectures and Microkernels.

It discusses the possibilities and limitation of plain virtualization approaches in embedded systems. These relate to the integrated nature of embedded systems with security and reliability requirements.

EXECUTIVE SUMMARY

Virtualization software is allowing systems developers to reduce hardware assets, or use them more efficiently, by running multiple "virtual machines" (VM) side by side on the same hardware, emulating different components of the systems. [1]

Each virtual machine operates almost as though it were a discrete physical host. This is achieved with a piece of software known as a hypervisor. The hypervisor is responsible for managing memory and Central Processor Unit (CPU) resources between the running virtual machines (also known as guest machines), providing a set of virtual hardware resources (such as display controllers, network interfaces, storage devices etc.) to guest hosts, and providing a control/management layer or channel between the system operator and the guest machines.

Virtualization has now moved beyond mainframes and servers and has found its way into applications, networks and storage, among others. [2] Users are beginning to reap a range of benefits from virtualization which go beyond the cost savings of hardware consolidation. For example, the military weapon upgrades, or reset programs which currently dominate government program allocations, trend toward reusing existing military platforms, whereas the theme is to upgrade existing platforms without compromising system integrity and reuse existing legacy proven code. These programs are using virtualization to not only cut hardware costs, but also to recover quickly from systems failures and maintain weapon systems or vehicle continuity.

A specific example is within a Tank Vehicle program with the US Army; new embedded devices have the capability to move an entire virtual machine instantaneously from one device to another and provide separation and isolation for vehicle security enclaves. Security, isolation or the building of on and off platform vehicle enclaves is performed through virtualization partitioning.

Benefits discovered include: hardware consolidation, reduced power and cooling (green computing), ease of deployment and administration, high availability and disaster recovery, migration and adoption of new hardware architectures, such as multi-core processors, processor obsolescence, semiconductor advances, etc., leverage/re-use legacy software code to meet existing safety and security-critical requirements and isolate GPL-licensed or other open source code from proprietary code or vehicle components.

Drawbacks of virtualization seem to be that users could face performance issues, as fewer systems do more work. Although virtualization technology lets systems operate fewer devices, it does not make programs run faster. There is no universal unique solution which may adequately solve each product's architectural goals or problems.

Virtualization is a technology which is modular in architecture, enabling developers to configure a custom product, specific solution that meets the required product-specific trade-offs between footprint, performance, isolation and security. Specific virtualization technology enables the full isolation of untrusted guest operating systems (OS) in hardware partitions. A structure decision can be applied when these factors are exposed and a specific need is applied across use-cases. The matrix provided within this document is an example of achieving a "best suited technology" to meet a specific need providing lowest risk to implement, as pertaining to virtualization.

INTRODUCTION

In computing, virtualization is a broad term that refers to the abstraction of computer resources. This paper refers to *Platform Virtualization*, which separates an OS from the underlying platform resources. Some terms which aid in the discussion and will be defined are; Full Virtualization, Hardware-assisted Virtualization, Partial Virtualization, Paravirtualization and Operating System-Level Virtualization. These technologies have evolved as an alternative or solution for emulation, and the *hypervisor* – a small segment of code designed to share physical resources between a number of logical virtual machines – is considered the most efficient way of doing it. [3] The hypervisor will take contended resources such as interrupt controllers and network cards, and present a synthetic version of them to each of the guest operating systems running in their own virtual machines (VM). A typical layered block diagram is shown in *Figure 1* with the hypervisor acting as *shim* or small piece of code managing or abstracting between hardware and operating system resources.

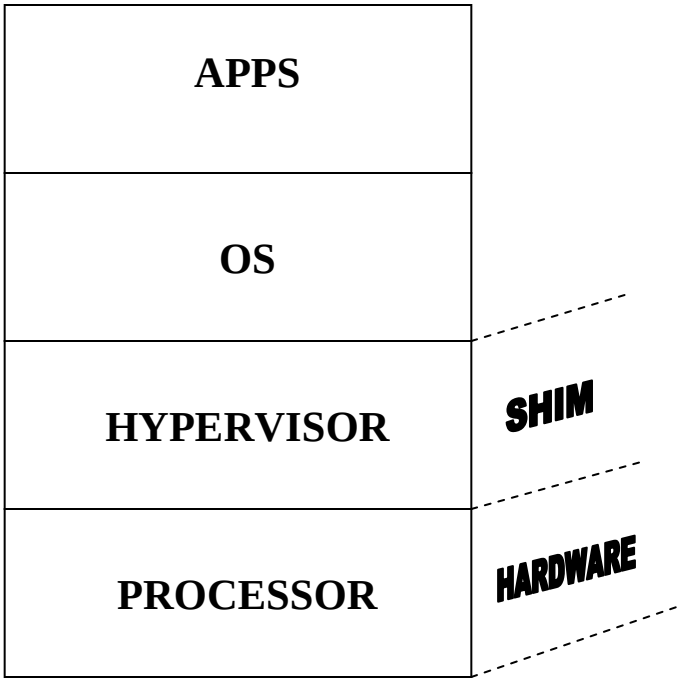


Figure 1: The Hypervisor Shim.

A basic cook book to perform virtualization consists of; Intel or Advanced Micro Devices (AMD) Hardware-assisted Virtualization Technology computer systems with a processor, chipset, BIOS, *Virtual Machine Monitor* (VMM) and for some uses, certain platform software enabled for virtualization. Functionality, performance, or other, benefit will vary depending on hardware and software configurations. Both the Intel or AMD Virtualization Technology-enabled BIOS and VMM applications together can be applied to fully take advantage of virtualization techniques. The different abstraction techniques consist of: *Platform* virtualization, which separates an operating system from the underlying platform resources; *Resource* virtualization, which is virtualization or emulation of specific system resources, such as storage volume, name spaces, and network resources; and *Application* virtualization, the hosting of individual applications on alien hardware/software. Pertaining to Platform Virtualization techniques; *Paravirtualization* is a virtualization technique that presents a software interface to virtual machines that is similar but not identical to that of the underlying hardware; *Full* virtualization is a virtualization technique which provides a complete simulation of the underlying hardware (Emulation). Another technique includes *Partial* virtualization, which provides partial simulation of the underlying hardware. Most, but not all, hardware features are simulated, yielding virtual machines in which some, but not all, software can be run without modification. *Operating System-Level* virtualization is a method where the kernel of an operating system allows for multiple isolated use-space instances of virtual machines, instead of just one. *Figure 2* represents the basic architecture of virtualization approach of partitioning operating environments and applications within instances of virtual machines managed by a virtual machine monitor (VMM or hypervisor) on a single hardware platform.

Virtualization Technology

Intel, with its VT technology, and AMD, with its AMD-V extensions to the x86 architecture, gave the *hypervisor* a privileged status, making it easier for the thin piece of code to adopt the role of supervisor, interpreting instructions from the guest operating systems without passing them blindly through to the processor. This becomes important if, for example, a guest operating system sends a ring zero-level command such as 'HLT', which would normally instruct the processor to be idle. In a virtualized environment, the processor will still probably be working on other tasks, for other systems, so the hypervisor must make this instruction local to their guest operating system so that it does not affect the whole computer. Also chips are now shipping with I/O support for native virtualization.

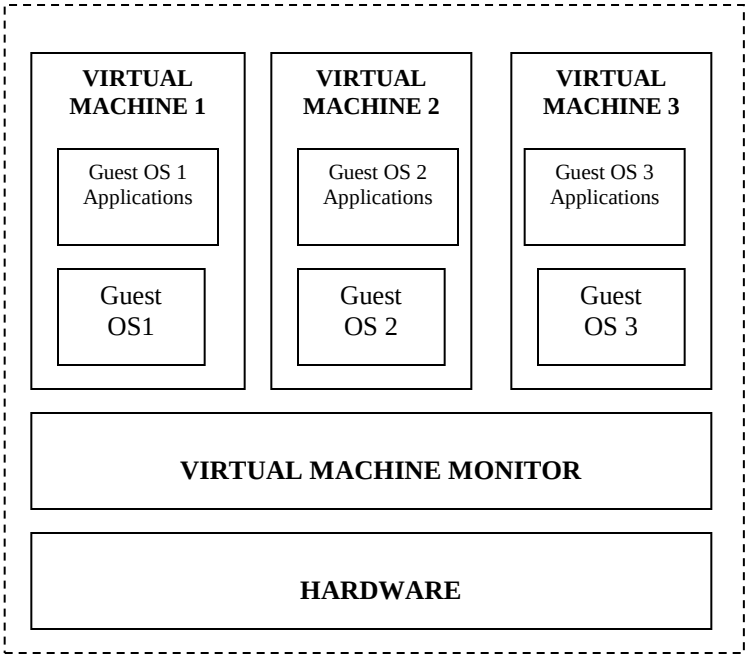


Figure 2: Example Virtualization Partitioning

HYPERVISOR

A *virtual machine monitor* (VMM) – Software running directly on the hardware that primarily functions as a host for one or more guest operating systems is referred to as a *Hypervisor*. It is described in two different types: [4]

Type 1, runs directly on the host's hardware to control the hardware and to monitor guest operating systems. This is also called *native or bare metal* type.

Type 2, runs within an operating system environment. This is also called *hosted*.

A hypervisor will handle interrupts from the operating system to the CPU, schedule CPU time among the guest operating systems and allocate cores to virtual machines, manage devices and allocate memory. Because an operating system does not speak directly to the CPU, the hypervisor must act as the intermediary when an operating system has an interrupt for the processor. When a guest wants to interrupt the processor, and when the processor has a response to that interrupt, the hypervisor must manage the delivery of those messages. Because guest operating systems compete for processor time, not all guests will be running all

the time. The hypervisor must store the response from an interrupt until the guest is operational again. The hypervisor must also coordinate the memory in which all of this takes place. Because guest operating systems do not know of each other, they will all assume that they have access to the same portions of physical memory.

Some disadvantages to using a hypervisor consist of Hypervisor's need to be monitored and managed once it is operating. This can be done with various levels of sophistication, but the "Holy Grail" is to have virtual machines started and stopped dynamically according to the system load or operating needs/requirements. Example of *Hypervisor Manager*: (virt-manager) http://virt-manager.org. Other issues; Increase in execution cycles; Increase code complexity; size; diminishing returns as number of VM increase; more points of failure.

ANALYSIS

A *paravirtualized* system typically performs better than a fully virtualized system because operations, – disk and network, have direct access to the hardware. However, it requires specially modified kernel and the kernel needs to be available for the OS you install. [5]

A *fully virtualized* system simulates all the hardware for the VM, permitting no direct access. It allows for a greater range of OS's, but generally will not perform as well as a paravirtualized system. Also for some hypervisor types, full virtualization requires hardware support, as well as possible acceleration support.

Use-cases generally fall into three broad main categories: [6]

1. *Co-existence* of different operating-system (OS) environments on the same platform,

2. *Isolating* critical components from an untrusted OS environment,

3. The use of an *Indirection* level for remote control of OS environments on deployed systems.

Modularity, *isolation* and *security* can be achieved once developers understand the required architecture need, the different choices of technology and the best use-case for applying the technology to a specific computing platform. Stronger inter-guest operating system isolation may be required to resist malware that may potentially be injected in one guest OS to penetrate another guest OS or application. To this end, virtualization utilizing a hypervisor module can provide complete isolation architecture between guest OS's. Isolation, in itself is not security, but only a prerequisite to the creation of a secure system.

LIMITATIONS

Within embedded systems design, virtualization efforts *may not* be most advantageous and careful consideration is warranted. Specifically existing software complexity may propose reliability concerns. Therefore, running legacy code as an objective, needs to be factored into the requirements. The highly integrated nature of embedded systems and their cooperating subsystems bring increase communication complexity and adding additional operating systems may prove less deterministic and introduce additional possible points of failure, thereby decreasing reliability. When designing a system with virtualization in mind, performance may actually decrease based on existing serialization or poor distribution of the software architecture. (i.e. separating cohesive units of software to isolate into separate processing entities) Adding too many virtual machines can have diminishing returns at some saturation level, based on the hardware specifications. Working through the different implementation techniques, such as with a structure decision matrix, should mitigate the risk and provide for better utilization of virtualization architectures.

MARKETS

The lists below continue to grow, but the market does contain some limited support and "not everything in the world has the capability to be virtualized".

Some vendors offering embedded virtualization solutions:

- Green Hills Software
- LynuxWorks
- Open Kernel Labs
- Real Time Systems GmbH
- SYSGO AG
- TenAsys
- VirtualLogix
- VMware (recently acquired TRANGO Virtual Processors)
- Wind River Systems

Popular Virtual Machine Software:

- ATL (A MTL Virtual Machine)
- Bochs, portable open source x86 and AMD64 PCs emulator
- CoLinux Open Source Linux inside Windows
- CoWare Virtual Platform
- Denali, uses paravirtualization of x86 for running paravirtualized PC operating systems.
- eVM Virtualization Platform for Windows by TenAsys
- FAUmachine
- Hercules emulator, free System/370, ESA/390, z/Mainframe

- KVM
- LilyVM is a lightweight virtual machine An introduction Logical Domains
- Microsoft Virtual PC and Microsoft Virtual Server
- OKL4 from Open Kernel Labs
- Oracle VM
- OVPsim is a freely available virtual platform simulator designed to simulate complex multiprocessor systems at very high speeds
- Parallels Workstation, provides virtualization of x86 for running unmodified PC operating systems
- Parallels Desktop for Mac, provides virtualization of x86 for running virtual machines on Mac OS X or higher
- QEMU is a simulator based on a virtual machine.
- SheepShaver.
- Simics
- Sun xVM
- SVISTA
- Trango Virtual Processors
- twoOStwo
- User-mode Linux
- VirtualBox
- Virtual Iron (Virtual Iron 3.1)
- VM from IBM
- VMware (ESX Server, Fusion, Virtual Server, Workstation, Player and ACE)
- vSMP Foundation (From ScaleMP)
- Xen (Opensource)
- IBM POWER SYSTEMS

Hardware with Virtual Machine support:

- Alcatel-Lucent 3B20D/3B21D emulated on commercial off-the-shelf computers with 3B2OE or 3B21E system
- AMD-V (formerly code-named Pacifica)
- ARM TrustZone
- Boston Circuits gCore (grid-on-chip) with 16 ARC 750D cores and Time-machine hardware virtualization module.
- Freescale PowerPC MPC8572 and MPC8641D
- IBM System/370, System/390, and zSeries mainframes
- Intel VT (formerly code-named Vanderpool)
- Sun Microsystems sun4v (UltraSPARC T1 and T2) -- utilized by Logical Domains
- HP vPAR and cell based nPAR
- GE Project MAC
- Honeywell Multics systems
- Honeywell 200/2000 systems Liberator
- IBM System/360 Model 145
- RCA Spectra/70 Series
- NAS CPUs emulated IBM and Amdahl machines
- Honeywell Level 6 minicomputers emulated predecessor 316/516/716 minis
- Xerox Sigma 6 CPUs

Other technologies can be used to accomplish virtualization use-cases rather than direct implementation of hardware-assistance from vendors. The next sections will briefly touch upon these concepts.

Hyper-Threading

Hyper-Threading (HT) is a means for improving processor performance by supporting the execution of multiple threads on the same processor at once: the threads share the various on-chip execution units. You can think of Hyper-Threading Technology as either a poor man's multiprocessor or a rich man's uniprocessor. [7] Some level of isolation, resource utilization can be attained, however mostly attained through application threads being properly separated or appointed to working with isolated CPU cores. Developers can take advantage of it either by multitasking (running a number of different applications at once) or by multi-threading (having multiple threads of control within an application).

CHIP ARCHITECTURES

Embedded devices are increasingly based on multi-core designs. Systems-on-a-chip (SoCs) often contain two or more processor cores in homogeneous or heterogeneous combinations, and FPGA-based designs can include a virtually unlimited number and variety of cores.

Traditional multiprocessing models have been one of two types. An asymmetric multiprocessing (*AMP*)-based RTOS is one approach to utilizing multi-core processors; symmetric multiprocessing (*SMP*) is another.

SMP systems are based on *homogeneous* hardware designs, where each CPU (or processor core) has identical capabilities and full access to all I/O devices and RAM in the system. SMP uses algorithms that perform dynamic load balancing by allocating software tasks among a number of identical processors to make maximum use of processor resources. A single operating system controls all of the processors. *Figure 3* represents the standard SMP architecture.

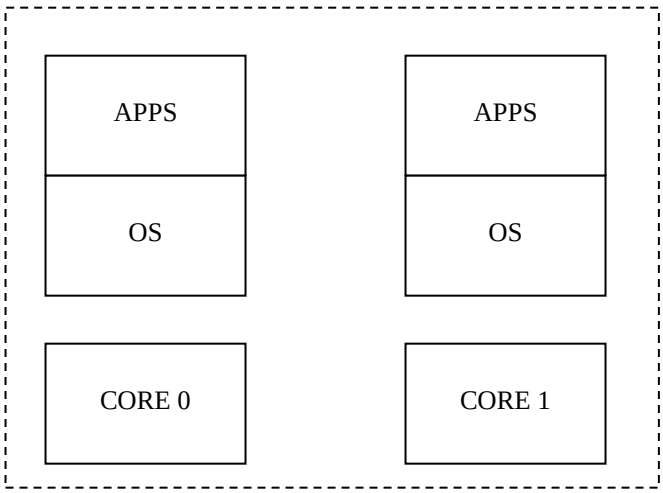


Figure 3: SMP

AMP systems are typically associated with *heterogeneous* hardware designs, where each CPU might have different features and capabilities and may even have dedicated I/O devices and RAM. AMP can best be described as "loosely-coupled" (or in some cases, completely uncoupled) multiprocessing. It performs selective load balancing; allowing the developer to permanently assign some tasks to fixed processor resources while allowing others to be load-balanced among many processors. There may be some communications among processors to exchange information and coordinate tasks, but there is no dynamic load balancing. The task assignment is fixed. *Figure 4* represents the standard AMP architecture.

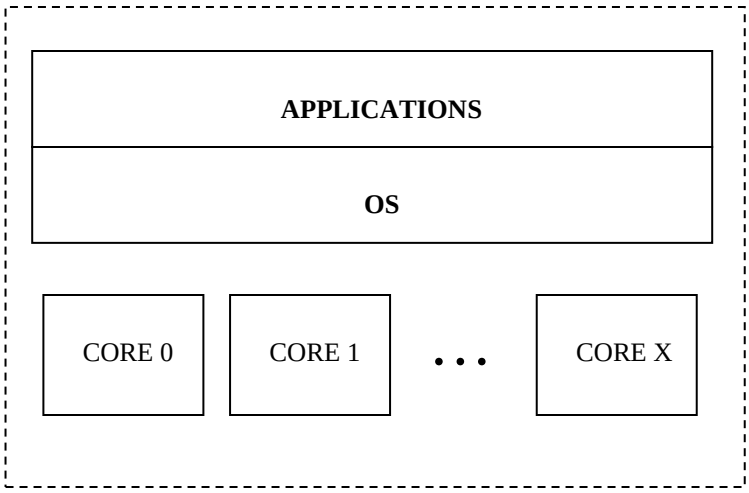


Figure 4: AMP

CPU ARCHITECTURE ANALYSIS

Dual-core and Quad-core machines by hardware architecture definition are SMP machines. The value of an SMP system is the ability to maximize the use of system resources. Having more than one CPU on which to schedule applications (or processes or threads or...) means less time waiting for high-priority CPU-intensive applications to finish. An AMP system is specializing through dedicated resources, where I/O devices and RAM are only accessible to a specific core and the applications that run on that core. This allows applications to assume sole ownership of the resources, with the benefit of less overhead and higher performance.

Virtualization through CPU Architecture

Hardware-virtualization technology, such as the Virtualization Technology (VT) found in many Intel dual-core and quad-core embedded processors, can be used for AMP configuration: to partition the CPU, RAM, and I/O devices of an SMP machine between multiple virtual machines. Unlike the server VMM model, the AMP-inspired embedded VMM requires multi-core processors and needs an assist from hardware-virtualization technology in the processor to ensure that each virtual machine has low interrupt latency, direct access to specialized I/O, and the assurance that the VMM will not "time slice away" the guest operating system and its applications.

Engineers contemplating a migration from a single-core to a multi-core processor must identify where parallelism exists in their application. The next decision is how to partition the code over the cores of the device. The two main options are symmetric-multiprocessor (SMP) mode and asymmetric-multiprocessor (AMP) mode, previously discussed. [8] In some cases, combinations of these make sense as well. There's just one kernel in SMP mode, and it's run by all cores. In AMP mode, each core has its own copy of a kernel, which could be different (heterogeneous operating systems) from, or identical (homogenous operating systems) to the one the other core is executing.

There are several factors that will guide the plan for the multi-core migration. Factors include the starting point (design) of the original source code, as well as migration goals and constraints. Each method has its own strengths.

More operating systems are now providing SMP, including embedded Real-Time Operating Systems (RTOS), but SMP requires code to be architected for parallelism to take advantage of multiple CPUs (parallelized). For situations where the application(s) is not well suited for parallelization, AMP and Virtualization could be a more viable solution for leveraging the extra processing capabilities of multi-core hardware. The ideal situation is to have SMP and AMP,

including virtualization, at your disposal. Amdahl's Law is in affect - which states that the upper limit on the speed-up gained by adding additional processors is determined by the amount of serial code that is contained in the application.

AMP Implementation

AMP requires no application changes to leverage the benefits of multiple cores. It can leverage multiple cores by running multiple instances of the OS and application in separate partitions that are dedicated to specific cores, Peripheral Component Interconnect (PCI) devices, and system memory areas. AMP requires a boot loader that supports AMP (can partition the hardware resources and make OS/application assignments to the partitions). The OS must also meet requirements to support AMP such as: The OS must be relocatable, must be able to restrict its memory region, and the OS must only operate on it assigned PCI devices.

Applying AMP to SMP Systems

Assigning resources exclusively to specific cores in an SMP system is a way to build an AMP system. By partitioning CPU cores, RAM, and I/O devices between multiple software systems, applications can gain direct control over the performance and use of those hardware resources. Running on a general purpose OS like Linux, AMP involves the use of interprocessor communication to combine the efforts of multiple processors, each with its own local operating system and hardware resources. [9]. AMP involves less OS overhead for each individual processor and a more traditional execution environment for applications.

SMP has more inherent code dependencies than AMP because the cores are more likely to contend for the same data as they execute similar code. However, AMP has its own issues with different cores sharing information, using interprocessor communication (IPC) and requiring semaphores. Sharing data can lead to data corruption and other parallel code challenges.

MICROKERNELS

Microkernel or *real-time operating system* (RTOS) is software that can serve as a host for one or more guest operating systems, but also serves as an operating system itself typically providing key features required by or desirable to the function of the target system (e.g., real-time performance, secure partitioning, and small footprint).

Why can't virtualization be performed simply with a Microkernel? Microkernel-based operating systems stripped down the privileged kernel part of the software, extrapolating many functions into separate components that can then be coordinated by the kernel. This creates a similar situation to the hypervisor, with a small '*shim*' that marshals communications between these components, and between the components and the processor. There are significant differences between the two. Microkernels are not suited to run meaningful applications, per its experts. Instead, they must be extended with higher level APIs to run UNIX-like applications. Hypervisors only know about the virtual machines and the guest operating systems they are managing, Microkernels take it upon themselves to handle tasks, threads, and memory contexts, impinging on the ground normally occupied by the guest operating system. Hypervisors are non-intrusive. You don't want to change the guest operating system. While Microkernels may not make good virtual machine managers, they are useful in situations where equipment needs to be nimble and quick to respond. [10]

SECURITY

For each system, a virtualized environment contains three extra 'layers' that may be attacked – physical host hardware, physical host OS and the hypervisor. If any of these are compromised, then all virtualized guest hosts on the physical system are compromised; also, as the attacker can then manipulate all aspects of guest hosts at will. Therefore, the physical and hypervisor layers should be closely guarded against unauthorized access.

Aside from attacks via the hypervisor, guest hosts are as vulnerable to direct attack as they would be as conventional physical systems. However, once a guest host is compromised, it is then possible to attack the hypervisor layer from the guest.

There is an element of communication between the hypervisor and the guest systems. This is made up of special communication channels, which allow client tools to communicate system state back to, or accept instructions from the management tool for the hypervisor and the operation of the various virtual hardware devices.

Bugs or back-doors in any of these components could be used to compromise the hypervisor or other guests running on the same hypervisor. Several such bugs have been discovered and demonstrated in existing virtualization software packages. [11]

Virtualization significantly weakens the security boundaries between objects in the same virtual domain. However, each of these issues can be managed with proper configuration and implementing the best suited technology for a use-case.

BEST SUITED TECHNOLOGIES FOR USAGE

Knowing there are several techniques and technologies to accomplish virtualization use-cases, a matrix can be developed to list a number of usages which relate to a possible virtualization need. Applying the analysis to items discovered during research, which are deemed better performing or more suited to use, one can show the best

suited technology for a specific usage. Associated hardware elements can also be placed within the matrix to show the processor architecture which can be applied for the usage and technology.

Listed below, derived from a number of sources and references, some possible use-cases for virtualization are shown:

- Virtual machines can be used to consolidate the workloads of several under-utilized systems to fewer systems.
- The need to run legacy applications. A legacy application might simply not run on newer hardware and/or operating systems.
- Provide secure, isolated sandboxes for running untrusted applications. Virtualization *is* an important concept in building secure computing platforms.
- Create operating systems, or execution environments with resource limits.
- Provide the illusion of hardware, or hardware configuration that you do not have (such as SCSI devices, multiple processors, ...)
- Simulate networks of independent computers.
- Run multiple operating systems simultaneously: different versions, or even entirely different systems.
- Debugging and performance monitoring. You can put such tools in the virtual machine monitor.
- Isolation for fault and error containment. You can inject faults proactively into software to study its subsequent behavior.
- Software migration, thus aiding application and system mobility.
- Create application suites as *appliances* by "packaging" and running each in a virtual machine.
- Tools for research and academic experiments. Since they provide isolation, they are safer to work with. They encapsulate the entire state of a running system: you can save the state, examine it, modify it, reload it, and so on. The state also provides an abstraction of the workload being run.
- Enable existing operating systems to run on shared memory multiprocessors.
- Create arbitrary test scenarios.
- Retrofit new features in existing operating systems without "too much" work.
- Effective means of providing binary compatibility.
- Co-locating hosts.

Putting together a matrix based on virtualization usage, technology and processor architecture, applying an analysis to point out "*best suited technology*" to utilize for a particular use-case, a structure decision can be invoked. The matrix in *Figure 5* is an attempt to structure the information in such a way.

Analysis Resources

Some of the resources used to perform the analysis, i.e. pin pointing what technique or technology is considered better or more defined, are listed below. Certain caveats may exist within the matrix as individual interpretations may propose opposition or justification for each of the technologies chosen as best suited for the usage.

Paul Fischer, TenAsys - Asymmetric real-time multiprocessing on multi-core CPUs. The latest multi-core processors are ideal for implementing multi-OS embedded applications. Virtualization technology makes it possible for a multi-core system to easily support multiple operating systems on a single computer platform.

Masaki Gondo, Director of Engineering, eSOL Co., Ltd - MULT I - C O R E DESIGN Information Quarterly Volume 5, Number 4, 2007 Blending Asymmetric and Symmetric Multiprocessing with a Single OS on ARM11 MPCoreM.

Gernot Heiser, OK Labs, Embedded systems virtualization: Consider a Hypervisor, By Full System Virtualization: Simulation for the Real-Time Embedded Economy.

Peter S. Magnusson, VIRTUTECH, October 2004, Host development platforms are now fast enough and hardware virtualization software is efficient enough to allow the development of OS and application code to take place before there is any real hardware–with advantages in cost and efficiency.

Michael Christofferson, Embedded.com, (11/09/05) Using an asymmetric multiprocessor model to build hybrid multicore designs.

Embedded.com (Jun 05 2007) SMP vs. AMP: How Homogenous Is Your Embedded System?

VMware - White Paper: Best Practices Using VMware Virtual SMP

Intel Technology Journal Hyper-Threading Technology Volume 06 Issue 01 Published February 14, 2002 ISSN 1535766X

Todd Brian, Embedded.com - Putting Multicore Processing in Context: Part 2 Dealing with hardware and OS issues (03/07/06)

Alan Murphy, F5 - White Paper – Technical marketing Manager, SECURITY1, Virtualization Defined – Eight Different Ways

Usage	Best-Suited Technology								Processor			Comments
	FULL VT	PARA VT	PARTIAL VT	OS VT	SMP	AMP	HYPER-THREADING	MICROKERNEL	PPC SINGLE CORE	INTEL SINGLE CORE	MULTI-CORE	
RESOURCE UTILIZATION	x	x	x	x	x	x	x	x	o	o	o	ALL provide for better utilization
SECURITY	x					x		x	o	o	o	pro/con need to be considered (x provides better support)
SEPARATION - TRUSTED/UNTRUSTED	x					x		x	o	o	o	
EMULATION OF RESOURCES	x	x	x				x		o	o	o	
ISOLATION	x											Complete OS
INCREASE PERFORMANCE		x			x	x	x	x	o	o	o	Application must be multithreaded by parallelization techniques, Para over Full for better performance
INCREASE THROUGHPUT					x						o	
RUN LEGACY CODE	x		x		x	x	x		o	o	o	
NETWORK SIMULATIONS	x								o	o	o	
OS- HETEROGENEOUS ENVIRONMENT	x					x			o	o	o	
OS-HOMOGENEOUS ENVIRONMENT		x			x	x			o	o	o	
DETERMINISTIC SYSTEMS	x					x		x	o	o	o	
SOFTWARE MIGRATION		x	x	x					o	o		
SEAMLESS RESOURCE SHARING					x			x			o	
SOFTWARE RETROFITTING	x	x	x	x	x				o	o		
ADMIN FUNCTIONS	x	x	x	x					o	o	o	
TESTING/TOOLS/DEBUGGING	x			x	x			x	o	o	o	
CO-LOCATING HOSTS	x	x	x	x					o	o	o	
BINARY COMPATIBILITY					x	x					o	
INTER-APP MESSAGING		x		x	x			x	o	o	o	
MULTI-THREADED APPLICATIONS					x		x		o	o	o	
HOTPLUG EXECUTION ENVIRONMENTS					x	x					o	Switch out to another system when failure occurs
LOW COST	x	x	x	x							o	
LOW COMPLEXITY	x						x	x	o	o		
REDUCED POWER					x	x					o	
SCALABILITY						x					o	
FAULT-TOLERANCE						x					o	

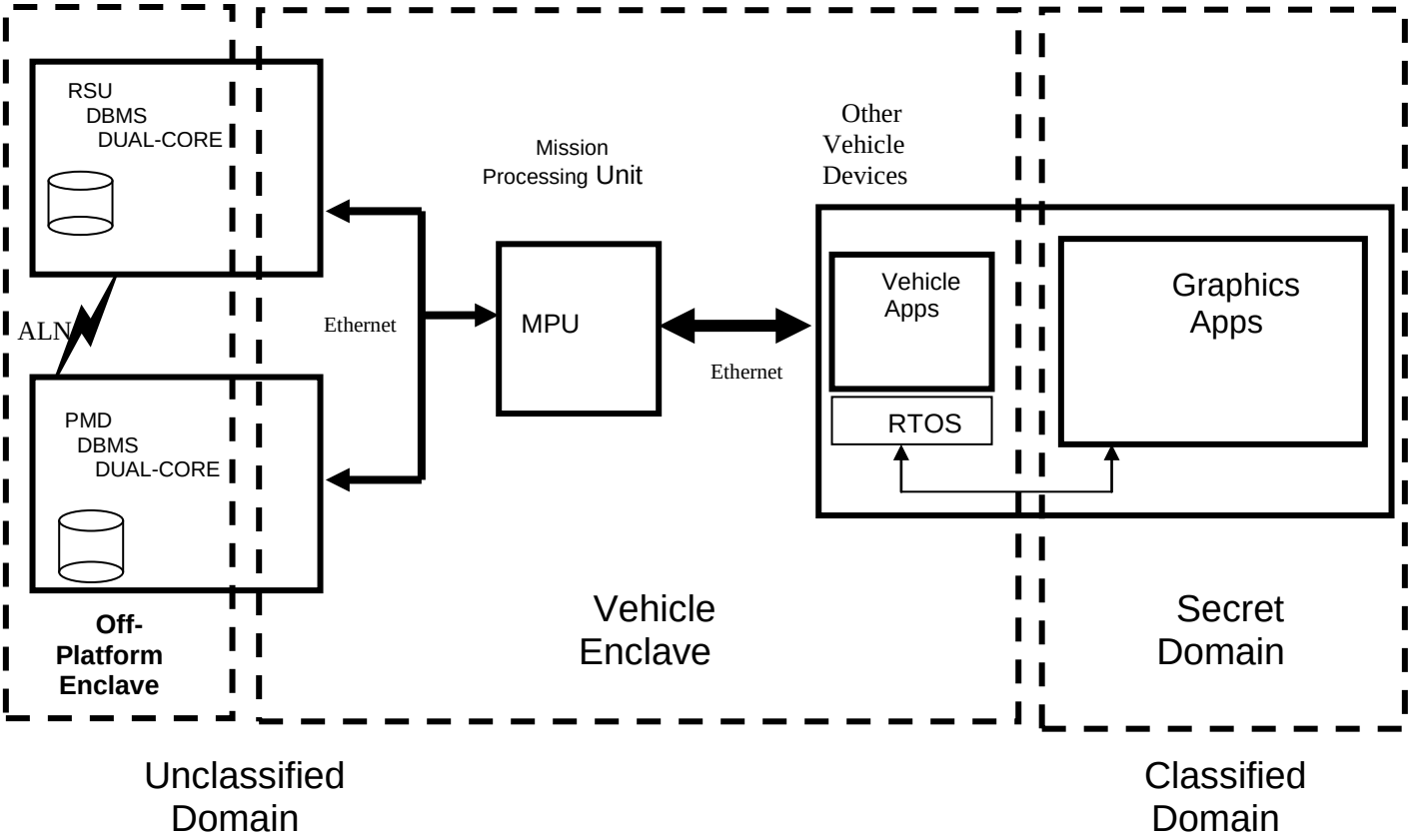
Figure 5: Usage-Technology-Architecture Matrix

Virtualization Opportunity

A specific example is within a Tank Vehicle program with the US Army; new embedded devices for vehicle training, simulation and diagnostic recording have the capability to move an entire virtual machine instantaneously from one device to another, and make use of virtual storage, applications and networking separation and isolation for vehicle security enclaves. The network is also virtualized, so the virtual machine performs an isolation of its network and connections.

Security, isolation or the building of on/off platform vehicle enclaves is performed through virtualization partitioning. This will protect the vehicle from outside unclassified networks and reduce the risk of inside (on-platform) classified networks or embedded systems processing classified data from intrusion or safety depreciation.

Some of the features virtualization can accomplish within this architecture: Security, OS Separation, Hardware Separation, Emulation of Resources (Network Interfaces), Dual-Core Systems (DSU, PMA), and Hardware Emulation. The goal is to achieve a communication path between enclaves other than directly through the physical Ethernet interface reserved for vehicle enclave. Other interests will be to emulate the hardware Ethernet interface on the Off-Platform enclave. Figure 6 represents a brief architecture diagram depicting the concept.



Legend:

RSU –	Recording Simulation Unit (Embedded Processing System)
PMD –	Portable Maintenance Device (Embedded Processing System)
ALN –	Army Logistical Networks (wireless)
DBMS –	Database Management System
MPU –	Mission Processing Unit (Embedded Processing System)
RTOS –	Real-Time Operating System

Figure 6: Example Design

STRUCTURED DECISION

The decision to utilize virtualization to accomplish memory/processor partitioning of DSU and PMD is a design for providing separation of DBMS and Wi-Fi from vehicle interfaces. Wi-Fi, DBMS can be isolated and the embedded devices are virtualization candidates, an opportunity to exploit the CPU cores for effective processing. The intent is to limit application and protocols to restricted user space to reduce risk and increase integrity of vehicle software infrastructure. Reviewing the *Virtualization Opportunity*, we can identify particular elements (objectives) and cross-associate them to the "Best Suited Technology" matrix. Once we have identified the usages, the best choice to achieve the required objectives stands out based on the column which identifies maximum coverage. Also the CPU Architecture is considered and identified within the matrix.

The clear choice and the Technology considered "best" to utilize for the usages identified within the design presented is to institute *Full Virtualization* and *Multi-core* is featured. *Figure 7* represents the cut-out, slice of the matrix identifying the structure decision.

Implementation utilizing the *Full Virtualization* techniques provides for core separation or pinning into virtual hardware machines running separate Operating Systems to establish virtual network adapters which provide further isolation from internal and external vehicle networks. The separation can be viewed as the security enclaves of the requirement objective. *Figure 8* represents an architecture view of the separation. (Same 'Legend' from Figure 6 applies)

Utilizing other virtualization techniques or technologies can accomplish the requirements or objectives. However, the least risk to fully institute an application based on the identified usages is presented. A structured decision based on analysis across a broad spectrum of technologies pertaining to virtualization could be part of the design and implementation process. Thus, would enable a system designer to fully deal with inherit risks when delivering embedded systems utilizing virtualization.

Usage	Best-Suited Technology								Processor		
	FULL VIRTUALIZATION	PARA VIRTUALIZATION	PARTIAL VIRTUALIZATION	OS VIRTUALIZATION	SMP IMPLEMENTATION	AMP IMPLEMENTATION	HYPER-THREADING	MICROKERNEL	POWERPC SINGLE CORE	INTEL SINGLE CORE	MULTI-CORE
SECURITY	x					x		x	o	o	o
SEPERATION	x	x	x	x		x		x	o	o	o
ISOLATION	x										
RUN LEGACY CODE	x	x	x		x	x	x		o	o	o
EMULATION OF RESOURCES	x	x	x				x		o	o	o

Figure 7: Selection from Usage-Technology-Architecture Matrix

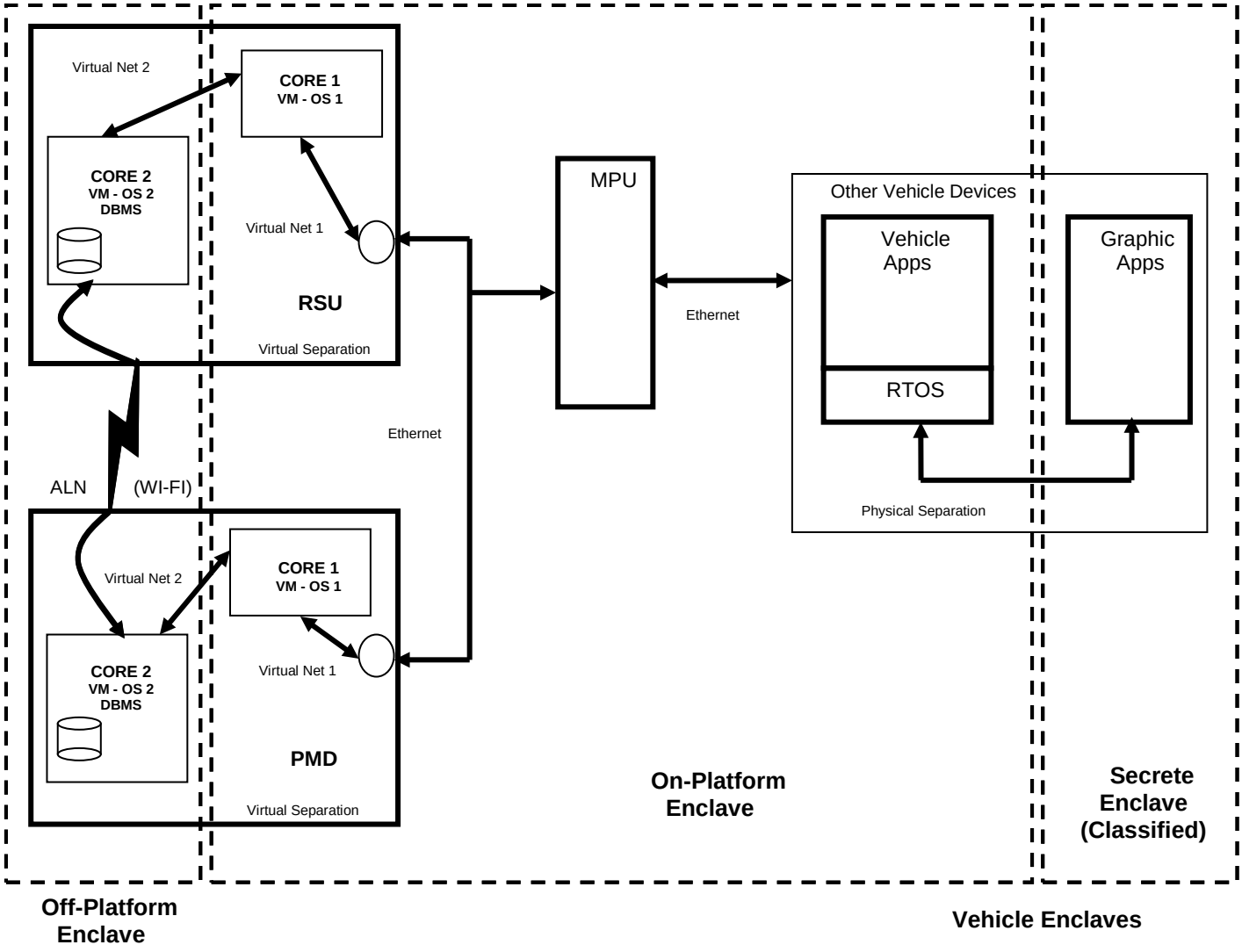


Figure 8: Isolation utilizing core pinning and network virtualization with Full Operating System separation with Virtual Machines

Conclusion

Embedded project requirements with objectives to preserve legacy investments, and with new software and hardware for integration may fit into development environments pertaining to virtualization. Finding the right technique without compromising existing solutions for safety or reliability is a component of knowing the power and limitations of a unique set of technologies. When working with a diverse set of use-cases, developing or categorizing the best suited technology against the choices can reduce or mitigate the risk to developing solutions as broad as technologies pertaining to virtualization.

Specifically, Virtualization is a technology which is modular in architecture enabling developers to configure a custom product, specific solution that meets the required product-specific trade-offs between footprint, performance, isolation, and security. Specific virtualization technology enables the full isolation of untrusted guest OS in hardware partitions. The modular architecture of virtualization technologies allows developers to make explicit trade-offs between the required level of isolation and the desired level of performances. A structure decision can be applied when these factors are exposed and specific needs are applied across use-cases. The matrix provided within this document is an example of achieving a best suited technology to utilize to meet a specific need at the lowest risk to implement, as pertaining to virtualization.

REFERENCES

[1] Arif Mohamed, “Virtualisation making IT more cost effective”, ComputerWeekly.com, article 221516, page 1, 2007.

[2] Arif Mohamed, “Virtualisation making IT more cost effective”, ComputerWeekly.com, article 221516, page 2, 2007.

[3] Danny Bradbury, “Everything you ever wanted to know about virtualisation”, ComputerWeekly.com, article 234352, page 1, 2009.

[4] IBM Systems, “Virtualization”, release 1, version 2, page 3, 2005.

[5] VMware, “Understanding Full Virtualization, Paravirtualization, and Hardware Assist”, whitepaper, revision: 20070911, item WP-028-PRD-01-01, page 5, 2007.

[6] Gernot Heiser, OK Labs, “Embedded systems virtualization: Consider a Hypervisor”, 212902574, page 1, 2009.

[7] Intel, “Hyper-Threading Technology: A Programmer’s Perspective”, page 1, 2003.

[8] Toby Fisher, “Symmetric Multiprocessing Vs. Asymmetric Processing”, Electronic Design, page 1, 2007.

[9] Toby Fisher, “SMP vs. AMP: How Homogenous Is your Embedded System?”, Embedded Technologies, page 1, 2007.

[10] Danny Bradbury, “Everything you ever wanted to know about virtualisation”, ComputerWeekly.com, article 234352, page 1, 2009.

[11] Rosenblum & Garfinkel, “Virtual Machine Introspection Architecture for Intrusion Detection”, February 2003.