DS 255 - System Virtualization
Assignment III - Constructs and Mechanisms for Virtualization

Shriram R.
M Tech (CDS)
06-02-01-10-51-18-1-15763

February 27, 2019

1. The two mainstream techniques for virtualizing a system are: System VMs and Process VMs. Some of the key differences between the two are described below:

   System Abstraction Layer - System VMs are realized in the ISA/HAL abstraction layer. Process VMs are realized in ABI (application binary interface) abstraction layer or API (Application programming interface) abstraction layer. The Process VMs realized in API layer are also known as HLL (High Level Language) VMs.

   Shared Resources - The system VMs share only the execution hardware which includes I/O devices, controllers, MMU, system interconnect (bus) and in some cases CPU cores. Note that some of these resources like CPU cores can be partitioned across VMs as well (i.e) dedicated to each VM.

   Process VMs in addition to execution hardware also share the same OS kernel (drivers + scheduler etc.) and possibly some libraries as well. This indicates that more resources are shared among Process VMs compared to System VMs.

   Performance - System VMs provide better performance as they are manifested in lower abstraction layer. Process VMs are however lightweight in terms of deployment but does not offer good performance compared to System VMs.

   Security - System VMs provide greater isolation between the guests thereby interference between them is minimized. However, vulnerabilities in the hypervisor can be exploited by the guest code to break the isolation.

   Process VMs are less secure compared to system VMs in terms of isolation since a fault due to user code in one process can potentially bring down the entire system through kernel panic thereby affecting other process VMs in the same system.

   To summarize, considering the goals of virtualization in terms of efficiency, isolation and safety: System VMs should generally satisfy isolation and safety goals better than Process VMs. Process VMs sometimes wins in terms of efficiency as it is lightweight.

2. Emulation is the process of implementing the interface and functionality of one system on another system having a different interface and functionality.

   It can be used to construct VMs for different ISA scenario by emulating one instruction set on hardware designed for another through techniques like interpretation and binary translation.

   Simulation is different from emulation in the sense that it tries to model the environment or functionalities of one system inside software with some approximations and use it run a software designed for that system.

   Simulation can be used to realize virtual machines although it comes with certain limitations. It can be used in evaluation or experimental study of new systems. Some of the key differences are given below,

Emulation	Simulation
Emulation can accurately implement a system's functionality and interface	Simulation is just a software model and so can potentially be inaccurate or approximate
Emulation can achieve better performance as it generally implemented in a lower abstraction layer and supported by hardware	Simulation has a negative impact on performance as it is generally run inside a computer program which is a higher abstraction layer
Emulation is rigid and quite challenging to continuously adapt to change in design	Simulation is a better tool in rapidly adapting to design changes as it is essentially a program

3. Virtualization is not exactly the same as abstraction but somewhat similar. Virtualization differs in the sense that it does not necessarily hide details (i.e) the level of detail in a virtual system is often same as that of the real system.

   Example: A Virtual machine can perform all kind of operations that it would perform on a real hardware. This is virtualization where the same level of interface / details is available to virtual machine.

   While the goal of abstraction is to hide the complexities of one layer in a system from another. However, both virtualization and abstraction provides an additional layer of interface between two existing layers in a system.

   Example: An hard disk is divided into sectors, tracks etc. But an application can read/write files in hard disk without the knowledge of structure of hard disk. This is abstraction where details are hidden.

4. **ISA**

   (a) Instruction Set Architecture (ISA) is the interface between hardware and software abstraction layer of a system

   (b) Application programs and operating system code interact with the hardware through ISA

   (c) It is generally divided into User and System ISAs targeted for user applications and OS operations respectively

   (d) All System VMs are defined in this layer

   ABI

   (a) Application Binary Interface (ABI) provides applications with user instructions (User ISA) and system call interface

   (b) System call interface is used to request privileged functions related to shared hardware etc.

   (c) Note that the system ISA is not exposed to applications in this interface

   (d) Process VMs are defined in this layer

   API

   (a) Application Programming Interface (API) provides access to user ISA and standard libraries

   (b) Applications can invoke various system services through the library

   (c) API provides an abstraction (sometimes a wrapper) to the implementation of these services

   (d) High Level Language (HLL) VMs are defined in this layer

5. The challenges in building virtualization support at hardware level are as follows,

   (a) Emulators for different ISAs especially CISC based are challenging to construct in hardware without any software support.

   (b) Hardware based ISA translation poses a limitation in code optimization capabilities. Techniques like inter-instruction optimization, batch optimization are difficult to directly implement in hardware level.

   (c) Making the hardware aware of the exact context of execution of processes in each virtual machines is one of the biggest challenge and prohibits higher efficiency

   (d) Design is rigid being at hardware level and so it is challenging to adopt changes to other higher abstraction layers and to add functionalities to support arbitrary guest OS or ISA

   (e) Virtualizing I/O devices is difficult since it requires additional support from hardware to export these as virtual I/O devices

   The benefits of having virtualization support built in hardware are as follows,

   (a) Hardware support can remove the need for para-virtualization and can avoid I/O emulation for direct-mapped I/O devices

   (b) It makes the VM state management simple by allowing the states to be stored in native structures during virtual machine execution

   (c) Hardware support for Interrupt virtualization in terms of handling external interrupts and interrupt masking greatly simplifies the complexity of VM design

   (d) Hardware based VMs can provide performance comparable to non-virtualized systems as it avoids the additional software layer for Hypervisor

   (e) It eliminates the need for shadow page table maintenance used by the hypervisor therefore saving memory. This also reduces memory translation overhead

6. The comparison of Interpretation, Binary translation and Para-virtualization in terms of different parameters is given below,

Parameter	**Interpretation**	**Binary Translation**	**Para-virtualization**
Legacy applications	Offers great support irrespective the source ISA (RISC or CISC). But not a good choice if the legacy code is unoptimized	Support depends on ISA and mitigation of problems like code discovery and location using incremental translation	Support is limited since it requires modification of guest code. Also, if the ISA is different then this technique cannot be used
Legacy runtime environments	Not recommended as optimization support is limited which is critical for runtime environments	Offers best support since runtime environment offers large scope for optimization which is one-time	Needs modification of guest. So, if the legacy env. is proprietary, then supported is highly limited
Application performance	Low translation cost but high execution cost and can affect performance since there is no code optimization and also higher overheads	High translation cost and low execution cost and can improve performance greater than interpretation if the code gets optimized	Overheads of interacting with guest device driver or guest software is reduced. Improves performance of the VM to better than Emulation techniques

7. **Different OS - Same ISA** - It is feasible to have para-virtualization. Despite being a different OS, the hypervisor can still modify the guest OS to trap the privileged instructions

   Same OS - Different ISA - It is not feasible to have para-virtualization if the ISAs are different. This is because para-virtualization on its own cannot perform emulation or binary translation to convert source to target ISA

   Different OS - Different ISA - It is not feasible to have para-virtualization in this case as well. The reason is same as above that the para-virtualization is designed to work on same ISA scenarios with modification to guest OS

8. Desirable characteristics in an ISA for Emulation are as follows,

   (a) Self-modifying and self-referencing code should not be permitted as it incurs additional challenges and overhead to emulation

   (b) Stack oriented memory architecture with an abstract memory model if infinite size is desirable as it allows for simple maintenance of program state

   (c) Indirect jumps should not be permitted as it will cause additional overhead to emulation in terms of finding the code location

   (d) Precise exception state requirement has to be relaxed and traps should be limited. This means that the exceptions are tested within the program

   (e) Variable length instructions, padding and mixing of code and data should not be permitted as it poses challenges in code discovery

References

1. Jim Smith and Ravi Nair - Virtual Machines: Versatile Platforms for Systems and Processes

2. Course Lecture Notes