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 25, 2019

1.

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. 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 continously 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. 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. 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.

4. **ISA** - Instruction Set Architecture (ISA) is the interface between hardware and software layer of a system. Application programs and operating system code interact with the hardware through ISA. It is generally divided into user and system ISAs targeted for user applications and OS operations respectively. All System VMs are defined in this layer.

ABI - Application Binary Interface (ABI) provides applications with user instructions (user ISA) and system call interface. The system call interface is used to request privileged functions related to shared hardware etc. Note that the system ISA is not exposed to applications in this interface. Process VMs are defined in this layer.

API - Application Programming Interface (API) provides access to user ISA and standard libraries. Applications can invoke various system services through the library. The API provides an abstraction (sometimes a wrapper) to the implementation of these services. High Level Language (HLL) VMs are defined in this layer.

5.

6.

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