Indian Institute Of Technology, Delhi


# COL733: Cloud Computing Technology Fundamentals

*Instructor: S. C. Gupta*

---

**Assignment 5: Design of Guest OS/Hyperviser**

---

***Report***

October 30, 2019

*Submitted To:*
S. C. Gupta
Professor
Computer Science Department

*Submitted By: (Group 4)*
Shantanu Verma 2016CS10373
Pradyumna Meena 2016CS10375
Manav Rao 2016CS10523
Shubham 2016CS10371

# Index

# 1. Need for redesigning the architecture

Current virtualization strategies are very unpredictable and the multifaceted nature of virtualization has been developing throughout the years just in light of the fact that the underlying computers were not built to keep virtualization in the mind. As we are advancing towards a further developed society there is a requirement for further developed, all the more computationally proficient, progressively vigorous frameworks which could utilize the resources they are furnished with. Subsequently, there emerge the requirement for better Cloud (virtualized) frameworks that could take into account our needs and yet are basic enough that they are viable to be utilized to achieve the tasks.

# 2. Proposed Architecture

Virtualization is accomplished in three components -

- Memory Virtualization
- I/O Virtualization
- CPU Virtualization

We will try to look at each component of virtualization, their entanglements and propose a few arrangements which may help in diminishing the complexities as well as virtualization overheads because of more seasoned design.

## 2.1. Memory Virtualization

The first and foremost change we propose in the Guest OS is correspondence to different OSes on the virtual machine by means of APIs like System Calls which should be implemented at the hypervisor level. Through these API calls, the OSes would know that they are not by any means the only OS present. We can also designate some predefined adequate lumps of memory to every one of the OSes at first yet expandable when required. This portion of memory would be like the memory allotment to forms in conventional frameworks where hypervisor holds some memory for itself and distributes the remainder of it to visitor OSes and visitor OSes can speak with one another like the procedures utilizing IPC (Inter-Process Communication) systems. The utilization of TLB can be changed accordingly to distribute visitor OSes as enormous procedures.

## 2.2. I/O Virtualization

The current method for implementing I/O virtualization is through three possible interfaces a) System Call b) Device Driver and c) I/O operational level with each level having their pros and cons. We here propose a generic virtual device driver interface. Every physical device that the hypervisor has is supplemented by a virtual device to the guest OS. The guest OSes will send the I/O requests to the virtual devices where they will all be entertained based on the priority and the type of request they've made. For eg. We

can give the spooling requests like Fax, Printing the least priority as they require contiguous time slices to complete the request without any breaks.

## 2.3. CPU Virtualization

The current virtualization techniques incur a lot of extra cost to cater to the sensitive and privileged instructions. Like the hardware solution puts the hypervisor at Level -1 instead of level 0 and all the sensitive instructions are trapped by hypervisor where as all the other instructions are trapped by the guest OS. Since the guest OSes are also not aware of the virtualization, all the instructions are first trapped to the guest OS.

In our solution, we make the guest OS's virtualization aware, as we proposed in the memory virtualization solution. Now, this way we can put the hypervisor at ring 0, guest OS at ring 1. Whenever there is a need to trap privileged/sensitive instruction the OS can itself figure whether the instructions should trap to guest OSes interrupt handler or hypervisors interrupt handler without always having to interrupt to guest OS interrupt handling vector. Hence saving a lot of computational time.