Assignment 4 : COL733

Design of Guest OS and Hypervisor

Group 20

Authors	Rachit Arora	Vaibhav Bhagee	Ankush Phulia	Kabir Chhabra
Entry No.	2014CS50292	2014CS50297	2014CS50279	2013CS50287

Introduction

We look at current virtualisation methods along the three major pillars : **Compute (CPU), Memory and I/O.** We then propose **fundamental design changes** to the current setting, i.e. We present an **alternate VMM and guest OS specification** that tackles some of the inefficiency faced in virtualisation currently.

The freedom to design an altogether new VMM and guest OS allows us to explore **new capabilities and options.** This was not possible in the traditional setting where the VMM and guest lack any sort of understanding about each other and have to work in a **safe but inefficient** fashion.

For instance;

- CPU virtualisation using para-virtualisation was not possible in legacy systems (systems without VT-x) because of the need to modify the guest OS.
- Memory virtualisation required shadow paging or extended page tables because VMM has no understanding of the memory needs of the guest OS system.
- Virtualising I/O at system call level was a challenge because of no knowledge of the specifics of guest system calls with the host.

A. CPU Virtualisation

A traditional operating system traps to ring level 0 whenever a system call is invoked or hardware event happens. User applications run at ring level 3.

Introducing virtualisation requires choosing a method to allow CPU virtualisation. An approach which traps all system calls to VMM rather than the guest OS (Trap and Emulate Model) is effective but inefficient. Even though hardware solutions like Intel VT-x exist to cater to this problem, it makes sense to design a system that eliminates this inefficiency even with legacy systems with no hardware support for CPU virtualisation.

The design approach of our VMM would be such that it would run at ring level 0 provided by the hardware. It would have the capability to handle hardware interrupts ar well as trap and perform sensitive instructions.

- The guest OS would run at ring level 1 with all system calls trapping from user (ring level 3) to level 1.
- Privileged instructions would trap to guest OS too.
- **However, sensitive instructions** would trap to **VMM (ring level 0).**
- This would be possible by writing a guest OS source code that explicitly traps to VMM for emulation of all sensitive instructions. Since we design **both VMM and guest OS** code from scratch, it is possible to achieve this.
- It is interesting to note that this approach is very similar to **para-virtualisation** with the only difference that guest OS code need not be modified and is written to **support trap to VMM on sensitive instructions beforehand**.

B. Memory Virtualisation

C. I/O Virtualisation