A BRIEF HISTORY OF VIRTUALIZATION

SIPB Cluedump Series 2010

Geoffrey Thomas

WHAT IS VIRTUALIZATION?

- Taking something not-virtual and making it virtual
  - Virtual memory
  - Virtual file systems
  - Virtual processors
  - Virtual machines

TYPES OF VIRTUALIZATION

- Application virtualization
- Desktop virtualization
- Platform virtualization
- Operating system virtualization

FORMAL DEFINITION

Gerald Popek (UCLA) and Robert Goldberg (Honeywell/Harvard), July 1974

"A virtual machine is taken to be an *efficient, isolated duplicate* of the real machine."

- Not a time-sharing operating system
- Not just virtual memory

FORMAL DEFINITION

- Efficiency: normal instructions must be executed directly
- Resource control: must trap to VMM when changing resource allocation
- Equivalence: A program (read: OS) must work the same with or without a VMM

FORMAL DEFINITION

- Privileged instructions: trap when called from user mode
- Sensitive instructions: either change resource allocation or are dependent on their physical-memory location or processor state
- Theorem: A VMM exists if sensitive ⊆ privileged

FORMAL DEFINITION

Does x86 virtualization work?

No, there are a bunch of unprivileged instructions that can read from processor state registers, or can impact physical-to-virtual memory.

X86 VIRTUALIZATION

So what now?

- Full processor emulation
- Dynamic translation (QEMU)
- Binary translation (Virtual PC, VMware)
- Paravirtualization (UML, Xen, VMware)
- Hardware support (KVM, VMware)

EMULATION

- Write a program that implements the processor.
- It works.
- It's slow.

DYNAMIC TRANSLATION

- Write a program that puts together code that implements the processor. Cache the results.

```
void op_movl_T0_r1(void)
{
        T0 = env->regs[1];
}
```

- It's a little faster.
- It can be easily ported.
- It's still not virtualization.

BINARY TRANSLATION

- Write a program to scan the code for sensitive instructions. Rewrite them. Cache the results.
- It's quite fast. It can be faster than actual hardware in some cases!
- Efficiency, resource control, and equivalence

PARAVIRTUALIZATION

- Give up on equivalence. Write your OS specifically for the VMM.

```
#ifdef CONFIG_XEN
#include <inc/xen/xen.h>
#include <kern/hypervisor.h>

void start_kernel(start_info_t *si)
{
    static char hello[] = "Bootstrapping...\n";

    (void)HYPERVISOR_console_io(CONSOLEIO_write, strlen(hello), hello);
    ...
```

PARAVIRTUALIZATION

- Give up on equivalence. Write your OS specifically for the VMM.
- Basically, make your OS a process in another OS.
- Porting is anywhere from annoying to impossible.
- Performance is in theory excellent.

HARDWARE VIRTUALIZATION

- Add support to your hardware to satisfy the Popek and Goldberg theorem.
- It's easy on the software writer end. It does require hardware support.
- Exits can be slow compared to either paravirtualization or clever BT.

OTHER APPROACHES

- VirtualBox
  - Run guest ring 0 code in host ring 1
  - Use some BT
  - Slightly sacrifices fidelity for speed.
  - x86-specific.

OTHER APPROACHES

- Paravirtualized drivers: “hypercalls” just for drivers
  - Paravirtualized block device (disks)
  - Paravirtualized network
  - Paravirtualized console
- Bootup and memory allocation still happens under BT. Drivers are easier to change than OS code.

OTHER APPROACHES

- Containers (OS virtualization)
- VMM : OS :: OS : application
- Add isolation capabilities until we reach equivalence
- “chroot but more awesome”

WHERE ARE WE GOING?

- Lines between platform/OS/virtualization blurred
- Use best techniques for performance
- Commonplace
- Cloud computing fulfills the promise of timesharing