Extended OS

Learning Outcomes

• An appreciation that the abstract interface to the system can be at different levels.
  – Virtual machine monitors (VMMs) provide a low-level interface
• An understanding of trap and emulate
• Knowledge of the difference between type 1 (native) and type 2 VMMs (hosted)

Virtual Machines

References:
All of chapter 7, if you’re interested.

Observations

• Operating systems provide well defined interfaces
  – Abstract hardware details
    • Simplify
    • Enable portability across hardware differences
• Hardware instruction set architectures are another well defined interface
  – Example AMD and Intel both implement (mostly) the same ISA
  – Software can run on both

Interface Levels

Instruction Set Architecture

• Interface between software and hardware
  – label 3 + 4
• Divided between privileged and unprivileged parts
  – Privileged a superset of the un-privileged
**Application Binary Interface**

- Interface between programs ↔ hardware + OS
  - Label 2+4
- Consists of system call interface + unprivileged ISA

**Application Programming Interface**

- Interface between high-level language ↔ libraries + hardware ↔ OS
- Consists of library calls + unprivileged ISA
  - Syscalls usually called through library.
- Portable via re-compilation to other systems supporting API
  - or dynamic linking

**Some Interface Goals**

- Support deploying software across all computing platforms.
  - E.g. software distribution across the Internet
- Provide a platform to securely share hardware resources.
  - E.g. cloud computing

**OS is an extended virtual machine**

- Multiplexes the “machine” between applications
  - Time sharing, multitasking, batching
- Provided a higher-level machine for
  - Ease of use
  - Portability
  - Efficiency
  - Security
  - Etc….

**Abstraction versus Virtualisation**

- (a) Abstraction
- (b) Virtualization

**Process versus System Virtual Machine**

- Guest
- Operating system
- Hardware
- Etc.

- Application process
- Interacting software
- System virtual machine
- Etc.
JAVA – Higher-level Virtual Machine

- write a program once, and run it anywhere
  - Architecture independent
  - Operating System independent
- Language itself was clean, robust, garbage collection
- Program compiled into bytecode
  - Interpreted or just-in-time compiled.
  - Lower than native performance

Comparing Conventional code execution versus Emulation/Translation

Aside: Just In-Time compilation (JIT)

JAVA and the Interface Goals

- Support deploying software across all computing platforms.
- Provide a platform to securely share hardware resources.

Issues

- Legacy applications
- No isolation nor resource management between applets
- Security
  - Trust JVM implementation? Trust underlying OS?
  - Performance compared to native?

Is the OS the “right” level of extended machine?

- Security
  - Trust the underlying OS?
- Legacy application and OSs
- Resource management of existing systems suitable for all applications?
  - Performance isolation?
- What about activities requiring “root” privileges
Virtual Machine Monitors

Also termed a *hypervisor*

- Provide scheduling and resource management
- Extended “machine” is the actual machine interface.

IBM VM/370

- CMS a light-weight, single-user OS
- VM/370 multiplex multiple copies of CMS

Advantages

- Legacy OSes (and applications)
- Legacy hardware
- Server consolidation
  - Cost saving
  - Power saving
- Server migration
- Concurrent OSes
  - Linux – Windows
  - Primary – Backup
  - High availability
- Test and Development
- Security
  - VMM (hopefully) small and correct
- Performance near bare hardware
  - For some applications

Native (Type 1) vs. Hosted (Type 2) Hypervisor

Type 1 (Native) Hypervisor

- Hypervisor (VMM) runs in most privileged mode of processor
  - Manage hardware directly
  - Also termed classic…, bare-metal…, native…
- Guest OS runs in non-privileged mode
  - Hypervisor implements a virtual kernel-mode/virtual user-mode
  - Hardware provides three privilege levels (e.g. Intel VT-x)
- What happens when guest OS executes native privileged instructions?

Type 2 (Hosted) Hypervisor

- Hypervisor runs as user-mode process above the privileged host OS
  - Also termed hosted hypervisor
- Again, provides a virtual kernel-mode and virtual user-mode
- Can leverage device support of existing host OS.
- What happens when guest OS execute privileged instructions?
Hosted Hypervisor Details

- Jeremy Sugerman, Ganesh Venkitachalam and Beng-Hong Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor", USENIX ATC 2001
- Hypervisor application installs driver (part of the hypervisor) into the Host OS
- Driver intercepts hypervisor related activities from Hyp. App.
- If "world switches" when guest OS needs to run:
  - Unloads Host OS state from processor
  - Loads hypervisor state and gives control of machine
- Hypervisor "world switches" when Host OS is needed:
  - Regularly to allow interactivity with Host OS.
  - When hypervisor needs Host OS service (e.g. file system)

Applications
Guest OS
Hypervisor
VM World
Host World

Physical Machine

Approach: Trap & Emulate?

Example: mtco/mfc0 MIPS

- mfc0: load a value in the system coprocessor
  - Can be used to observer processor configuration
- mtco: store a value in the system coprocessor
  - Can be used to change processor configuration

Example: disable interrupts

- mfc0 r1, C0_Status
- andi r1, r1, CST_IEc
- mtco r1, C0_Status
- Sensitive?
- Privileged?

Example: cli/sti x86

- CLI: clear interrupt flag
  - Disable interrupts
- STI: set interrupt flags
  - Enable interrupts
- Sensitive?
- Privileged?

X86 POPF

- Pop top of stack and store in EFLAGS register
  - IF bit disables interrupts
X86 POPF

- Is not privileged (does not trap)
  - In kernel mode – enable/disables interrupts
  - In user-mode – silently ignored
- POPF is not virtualisable
- X86 (pre VT extensions) is not virtualisable

Taxonomy of Virtual Machines

What is System/161?