


CSE


Extended OS



CSE

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)



CSE

Virtual Machines


References:
 Smith, J.E.; Ravi Nair; , "The architecture of virtual machines,"
Computer , vol.38, no.5, pp. 32- 38, May 2005
 Chapter 7 – 7.3 Textbook "Modern Operating Systems", 4th ed.
 All of chapter 7, if you're interested.



CSE

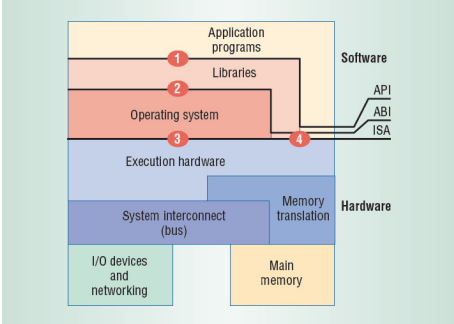
Observations

- 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




CSE

Interface Levels



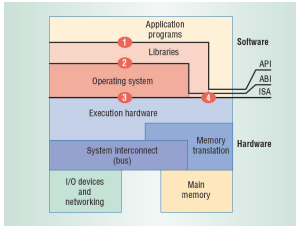
The diagram illustrates the layers of a system and the interfaces between them. The top layer is 'Software', which includes 'Application programs', 'Libraries', and 'Operating system'. The bottom layer is 'Hardware', which includes 'Execution hardware', 'System interconnect (bus)', 'Memory translation', 'I/O devices and networking', and 'Main memory'. Four numbered interfaces are indicated: 1 (between Application programs and Libraries), 2 (between Libraries and Operating system), 3 (between Operating system and Execution hardware), and 4 (between Execution hardware and System interconnect (bus)).




CSE

Instruction Set Architecture

- Interface between software and hardware
 - label 3 + 4
- Divided between privileged and un-privileged parts
 - Privileged a superset of the un-privileged



This diagram is identical to the 'Interface Levels' diagram, showing the layers of software and hardware and the interfaces between them. It highlights the interface between the operating system and the hardware (labeled 3 and 4) as the Instruction Set Architecture (ISA).



Application Binary Interface

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

Application Programming Interface

- Interface between high-level language ↔ libraries + hardware + OS
- Consists of library calls + un-privileged 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

Process versus System Virtual Machine

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

```

    graph TD
      A[Java Code (.java)] --> B((JAVAC compiler))
      B --> C[Byte Code (.class)]
      C --> D1[JVM]
      C --> D2[JVM]
      C --> D3[JVM]
      D1 --> E1[Windows]
      D2 --> E2[Linux]
      D3 --> E3[Mac]
    
```

Comparing Conventional code execution versus Emulation/Translation

(a) Conventional code execution: HLL program → Compiler front end → Intermediate code → Compiler back end → Object code → Loader → Memory Image.

(b) Emulation/Translation: HLL program → Compiler → Portable code → VM loader → Virtual memory image → VM interpreter/compiler → Host instructions.

Note: 'Distribution' is indicated between 'Object code' and 'Virtual memory image'.

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

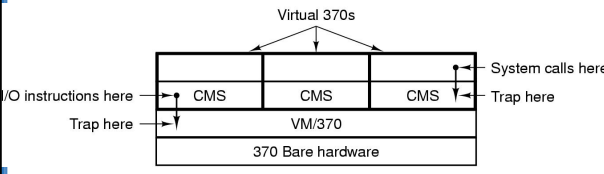

Also termed a *hypervisor*

- 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

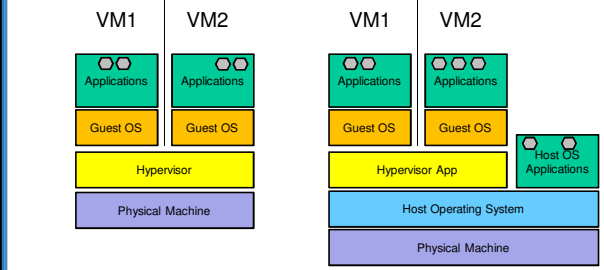




Advantages

- 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

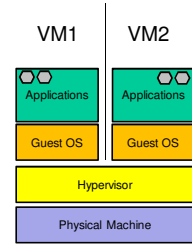



Native (Type 1) vs. Hosted (Type 2) Hypervisor

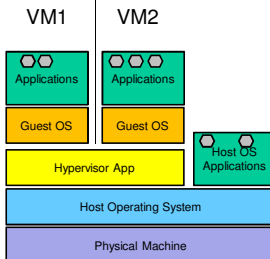

Type 1 (Native) Hypervisor

- 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.
- It "world switches" when guest OS needs to run
 - Unloads Host OS state from processor
 - Loads hypervisor state and gives it 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)

The diagram illustrates the layered architecture of a hosted hypervisor. It is divided into two main worlds: the Host World and the VM World, both running on top of the Physical Machine. In the Host World, there is a Host OS layer containing Host OS Applications and a Hyp. Driver. In the VM World, there is a Guest OS layer containing Applications and the Hypervisor. A Hypervisor App is shown as a separate component that interacts with the Host OS. The Physical Machine is the hardware foundation at the bottom.

Gerald J. Popek and Robert P. Goldberg (1974). "Formal Requirements for Virtualizable Third Generation Architectures". Communications of the ACM 17 (7): 412-421.

- Sensitive Instructions
 - The instructions that attempt to change the configuration of the processor.
 - The instructions whose behaviour or result depends on the configuration of the processor.
- Privileged Instructions
 - Instructions that trap if the processor is in user mode and do not trap if it is in system mode.
- Theorem
 - Architecture is virtualisable if sensitive instructions are a subset of privileged instructions.

Approach: Trap & Emulate?

The diagram shows two virtual machines, VM1 and VM2, separated by a vertical line. VM1 has a register labeled WB and another labeled ST. VM2 has a register labeled ST. Below VM2, it says EPC = bad address. There are also some handwritten notes like 'id = r1, ST' and 'WB M2'.

Example: mtc0/mfc0 MIPS

- mfc0: load a value in the system coprocessor
 - Can be used to observe processor configuration
- mtc0: 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
mtc0 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


| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | I | D | V | I | V | A | V | R | F | 0 | N | O | P | L | O | F | F | I | T | S | Z | F | 0 | A | F | 0 | P | F | 1 | C | F |

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

cse

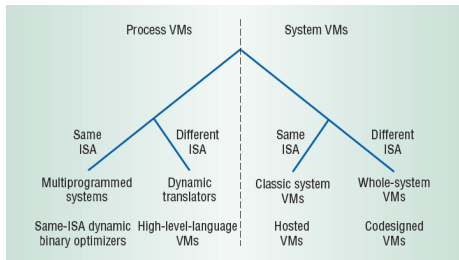
X86 POPF

- 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




cse

Taxonomy of Virtual Machines



```

graph TD
    Root[ ] --- Process[Process VMs]
    Root --- System[System VMs]
    Process --- P_Same[Same ISA]
    Process --- P_Diff[Different ISA]
    System --- S_Same[Same ISA]
    System --- S_Diff[Different ISA]
    P_Same --- P_Same_1[Multiprogrammed systems]
    P_Same --- P_Same_2[Same-ISA dynamic binary optimizers]
    P_Diff --- P_Diff_1[Dynamic translators]
    P_Diff --- P_Diff_2[High-level-language VMs]
    S_Same --- S_Same_1[Classic system VMs]
    S_Same --- S_Same_2[Hosted VMs]
    S_Diff --- S_Diff_1[Whole-system VMs]
    S_Diff --- S_Diff_2[Codesigned VMs]
    
```



cse

What is System/161?

