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 and type 2 VMMs
• An appreciation of some of the issues in virtualising the R3000
Virtual Machines

References:
Chapter 8.3 Textbook “Modern Operating Systems”
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

1. Application programs
2. Libraries
3. Operating system
4. Execution hardware

- System interconnect (bus)
- Memory translation
- I/O devices and networking
- Main memory

Software
- API
- ABI
- ISA

Hardware
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) 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
Comparing Conventional versus Emulation/Translation

(a) HLL program → Compiler front end → Intermediate code → Compiler back end → Object code → Loader → Memory image

(b) HLL program → Compiler → Portable code → Distribution → VM loader → Virtual memory image → VM interpreter/compiler → Host instructions
Aside: Just In-Time compilation (JIT)

\[
\begin{align*}
&- \text{main}(c) \leq b \\
&- \text{foo}(d) < de
\end{align*}
\]
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

• 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
Taxonomy of Virtual Machines

- Process VMs
  - Same ISA
    - Multiprogrammed systems
  - Different ISA
    - Dynamic translators
      - Same-ISA dynamic binary optimizers
      - High-level-language VMs

- System VMs
  - Same ISA
    - Classic system VMs
  - Different ISA
    - Whole-system VMs
    - Hosted VMs
What is System/161?
Figure 1-29. (a) A type 1 hypervisor. (b) A type 2 hypervisor.
Type 1 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
- What happens when guest OS executes native privileged instructions?
Type 2 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?

- **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?
Virtual R3000???

- Interpret
  - System/161
    - slow
  - JIT dynamic compilation

- Run on the real hardware??
Issues

- Privileged registers (CP0)
- Privileged instructions
- Address Spaces
- Exceptions (including syscalls, interrupts)
- Devices
mfcO r1, CO - Cassse
R3000 Virtual Memory Addressing

- MMU
  - address translation in hardware
  - management of translation is software

Figure 2.10 Virtual Memory Addressing
R3000 Address Space Layout

- **kuseg:**
  - 2 gigabytes
  - MMU translated
  - Cacheable
  - user-mode and kernel mode accessible
R3000 Address Space Layout

- **kseg0:**
  - 512 megabytes
  - Fixed translation window to physical memory
    - 0x80000000 - 0x9fffffff virtual = 0x00000000 - 0x1fffffff physical
    - MMU not used
  - Cacheable
  - Only kernel-mode accessible
  - Usually where the kernel code is placed
R3000 Address Space Layout

- **kseg1:**
  - 512 megabytes
  - Fixed translation window to physical memory
    - $0xa0000000 - 0xbfffffff$ virtual = $0x00000000 - 0x1fffffff$ physical
    - MMU not used
  - **NOT** cacheable
  - Only kernel-mode accessible
  - Where devices are accessed (and boot ROM)
R3000 Address Space Layout

- **kseg2:**
  - 1024 megabytes
  - MMU translated
  - Cacheable
  - Only kernel-mode accessible