Review: What is an OS?

- Software between applications and (ugly) hardware:
  - Abstracts hardware and makes portable, usable and pretty.
  - Makes finite into (near)infinite.
  - Provides protection.

How is a process different from HW?

<table>
<thead>
<tr>
<th>Process</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU – Non-Privileged registers and instructions.</td>
<td>CPU – All registers and instructions.</td>
</tr>
<tr>
<td>Memory – Virtual memory.</td>
<td>Memory – Both virtual and physical memory, memory management, TLB/page tables, etc.</td>
</tr>
<tr>
<td>Exceptions – signals, errors.</td>
<td>Exceptions – Trap architecture, interrupts, etc.</td>
</tr>
</tbody>
</table>

One way: Complete Machine Simulation

- Build a simulation of all the hardware.
  - CPU – A loop that fetch an instruction, decode it, simulate its effect on the machine, state.
  - Memory – Physical memory is just an array, simulate the MMU on all memory accesses.
  - I/O – Simulate I/O devices, programmed I/O, DMA, interrupts.
- Problem: Too slow!
  - 100x slowdown makes it not too useful.
  - CPU/Memory – 100x CPU/MMU simulation.
  - I/O Device – >2x slowdown.
- Need to emulate CPU/MMU fast enough.

Dynamic Binary Translation

- Problem: simulated CPU, MMU not fast enough
  - slow interpreter!
- Solution: translate & execute translated code

```assembly
lea r4, REG BASE
lw r2, R2_OFS(r4)
lw r3, R3_OFS(r4)
add r1, r2, r3
jal translate_r1
lw r1, 8(r1)
sw r1, R4_OFS(r4)
```
- Similar to just-in-time (JIT) compilation (e.g. Java, .net)
- ~4-10x slowdown (better if we can use hardware MMU?)
- Can we do better?

Making a process look like hardware - CPU

- Observations: Most instructions are the same regardless of processor privilege level.
  - Example: ```lns 1eax```
- Why not just let CPU execute the instructions directly?
  - Safety – How we going to get it back? Or stop it from stepping on us? How about CLI/HALT?
  - Answer: Use protection mechanism.
- Run virtual machine directly on CPU at non-privileged level.
  - Most instructions just work.
  - Privileged instructions trap into monitor and run simulator on instruction.
  - Makes some assumptions about architecture.
CPU Trap architecture virtualization

- What happens when an interrupt or trap occurs.
  Like all OSes: we trap into the monitor.
- What if the interrupt or trap should go to the VM?
  Example: Page fault, illegal instruction, system call, interrupt.
- Run the simulator again.
  X86 example: Lookup trap vector in VM's IDT.
    Push cs, eip, eflags, on stack.
    Switch to privileged mode.

Possible solutions (w/o HW support)

- Modify the OS? (Xen, User-Mode Linux approach)
  Change so it doesn't use non-virtualizable instructions.
    In fact, can just call VMM directly – possibly more efficient!
  Not general solution – can’t run Windows.
- Partial binary translation (VMware approach)
  Run user code directly.
    Translate kernel code, patching non-virtualizable instructions
    can also patch other instructions for performance!
  As with DBT, need to track self-modifying code.
    Map non-writable in MMU, trap & re-translate.
    Windows does this. Also pages share code & data.
  Have interpreter for code that is rarely used
    avoid translation overhead

MMU Virtualization

- Trick: Monitor keeps shadow of VM’s page table
  Contains mapping to physical memory allocated for that VM.
  Access causes Page Fault:
    Lookup in VM’s page table mapping from VPN to PPN.
    Determine where PPN is in machine memory (MPN).
    Insert mapping from VPN->MPN into shadow page table.
- Uses hardware for protection/isolation
  Monitor never maps itself into VM’s page table
  Monitor never maps memory allocated to other VMs in VM’s page table
- AMD’s Nested Page Tables

CPU Virtualization Requirements

- Need protection levels to run VMs and monitors
- All unsafe/privileged operations should trap
  Example: disable interrupt, access I/O dev, ...
  x86 problem: POPF (different semantics in different rings)
- Privilege level should not be visible to software
  Software in VM should be able to query and find its in a VM
  x86 problem: MOV ax, cs
- Trap should be transparent to software in VM
  Software in VM should be able to tell if instruction trapped.
  x86 problem: traps can destroy machine state.
- Lost art
  Re-found - Intel’s VT, AMD-V

Virtualization Requirements - Virtualizing Memory

Basic MMU functionality used by OS:
- OS manages physical memory (0…MAX_MEM).
- OS sets up page tables mapping VA->PA.
- CPU accesses VA to should go to PA. Paging off: PA=VA.
  Used for every instruction fetch, load, or store.
Need to implement a virtual physical memory
- Logically need additional level of indirection
  VM’s VA -> VM’s PA -> machine address
Trick: Use hardware MMU to simulate virtual MMU.
  Can be folded into page tables: VA->machine address

I/O device virtualization

- Type of communication:
  Special instruction – IN/OUT.
  Memory mapped I/O (PIO).
  Interrupts.
  DMA.
- Virtualization
  Make IN/OUT and PIO trap into monitor.
  Run simulation of I/O device.
- Simulation:
  Interrupt – Tell CPU simulator to generate interrupt.
  DMA – Copy data to/from physical memory of virtual machine.
Virtual Machine Monitor

- Thin layer of software that virtualizes the hardware
  - Exports a virtual machine abstraction that looks like the hardware.

![Diagram of Virtual Machine Monitor](image)

Old idea from the 1960s

- IBM VM/370 – A VMM for IBM mainframes
  - Multiplex multiple OS environments on expensive hardware.
  - Desirable when few machines around.
  - Also: make single-user OS “multi-user.”
- Interest died out in the 1980s and 1990s.
  - Hardware got cheap (with faster CPUs, huge mem/storage.)
  - Compare Windows XP vs. NDOS machines
    - However, Windows includes a DOS virtual environment. ;-) 
- Interesting again today
  - Different problems today – software management
  - VMM attributes still relevant

Virtual Machine Monitor attributes

- Software compatibility
  - Runs pretty much all software
  - Trick: Make virtual hardware match real hardware.
- Low overheads/High performance
  - Near “raw” machine performance.
  - Direct execution on CPU/MMU.
- Complete isolation
  - Total data isolation between virtual machines.
  - Use hardware protection.
- Encapsulation
  - Virtual machines are not tied to physical machines.
  - Checkpoint/Migration.

Different thinking about OSes

- Installed software on hardware is broken
  - Tight coupling of OS and applications to hardware creates management problems.
- Want to subdivide OS:
  - Hardware drivers.
  - Hardware management.
  - System support software.
- Turn OSes into normal software that can be managed

Backward compatibility with VMMs

- Backward compatibility is bane of new OSes.
  - Huge effort require to innovate but not break.
- Security considerations may make it impossible
  - Choice: Close security hole and break apps or be insecure
- Example: Not all XP applications run on Vista.
  - In spite of a huge effort to make Vista compatible.
  - Given the number of applications that run on XP, practically any change will break something.
  - If (OS == XP)...
- Solution: Use a VMM to run both XP and Vista
  - Obvious for OS migration as well: Windows -> Linux
  - keep running legacy apps, OS until you can replace them on new OS

Isolation: access to classified networks

- Traditional tension: Security vs. Usability
  - Secure systems tend not to be that usable.
  - Flexible systems are not that secure.
- Additional information assurance requirement:
  - Data cannot flow between networks of different classification.
- Solution: Run two VMs:
  - Classified VM
  - Internet VM
- Use isolation property to isolate two VMs
  - VMM has control of the information flow between machines
  - Declassifier mechanism
Logical partitioning of server machines

- Run multiple servers on same box
  - Ability to give away less than one machine.
  - Modern CPUs more power than most services need.
- 0.1U rack space machine - Better power, cooling, floor space, etc.
- Server consolidation trend: N machine -> 1 real machine.
- Isolation of environments
  - Printer server doesn’t take down Exchange server.
  - Compromise of one VM can’t get at data of others.
- Resource management
  - Provide service-level agreements.
- Heterogeneous environments
  - Linux, FreeBSD, Windows, etc.

Example: Using VMM to enhance security

- Trade-offs
  - Host-based IDS (HIDS):
    - + Good visibility to catch intruder.
    - - Weak isolation from intruder disabling/masking IDS.
  - Network-based IDS (NIDS):
    - + Good isolation from attack from intruder.
    - - Weak visibility can allow intruder to slip by unnoticed.
- Would like visibility of HIDS with isolation of NIDS.
  - Idea: Do it in the virtual machine monitor.

VMM-based Intrusion Detection System

- Strong isolation
  - VMM isolate software in VM from VMM.
  - Comprise OS in VM can’t disable IDS in VMM.
- Introspection – Peer inside at software running in VM
  - VMM can see: Physical memory, registers, I/O device state, etc.
  - Signature scan of memory
    - Look through physical memory for patterns or signs of break-in
- Interposition – Modify VM abstraction to enhance security
  - Memory Access Enforcer
    - Interpose on page protection.
  - NIC Access Enforcer
    - Interpose on virtual network device.

Virtual Appliances

- Virtualization decouples software from hardware
  - OS no longer an extension of hardware
- OS is bundled with application
  - Choose OS based on the needs of the application
  - only include what you need
- No additional installation required
  - VM includes OS, libraries, application, support software
  - No hardware incompatibilities (runs on virtual hardware)
  - No software incompatibilities (includes entire environment)
- “Virtual Appliance”
  - web server, mail server, standard desktop, test environment...
  - tons of them available on internet

Mobility

- Virtual machines allow you to push nodes around on your network
  - as long as network can route/switch seamlessly
- Move servers around to optimize for
  - performance (few VMs per machine)
  - power, hardware resources (many VMs per machine)
  - network latency (put VMs close to clients)
- Keep your work on a VM, and take it with you!
  - beyond afs home directory – get your whole machine
  - log into a machine – loads your VM
  - put it on a flash drive, send it over the network...

Live upgrades

- Can upgrade hardware, VMM while OS and applications continue to run
  - move VMs off machine you’re upgrading
  - upgrade/replace hardware and/or VMM
  - reboot and restart VMM
  - move VMs back
- Virtual machines make the OS much more flexible and useful, provide improved compatibility and isolation, and allow you to fix problems and limitations of a legacy OS!
## Question: Has OS design failed?

- **Microkernels**
  
  "A virtual machine monitor is a µkernel with a lousy API."
  
  Microkernels at same layer as VMM. Never caught on – why? Performance, compatibility.

- **Distributed (Operating) Systems**
  
  Merge systems over network into coherent whole. Provide process migration, unified file system, etc. Problems: performance, reliability, compatibility. Future: IFC, network security, Web/Cloud/Grid computing?

## Has OS design failed?

- **OS libraries and software environments**
  
  Poor support for multiple (e.g. legacy) environments. Destructive software installation. Poor compatibility.

- **Security: OS design (and implementation) has failed!**
  
  Buy new laptop + connect to internet => pwned. Viruses, worms, spyware, rootkits, spam, phishing, botnets, remote exploits, local privilege escalation, buffer overflows, TOCTTOU bugs, bad permissions, excessive privileges...
  
  Result: poor isolation means people are (rightly) afraid to run e-mail and web server in same OS.

- **VMMs** offer a general workaround to OS deficiencies, by implementing an ugly (hardware) interface which OSes were supposed to make pretty and useful!

---

## The Operating System

### Traditional View

- **OS jobs:**
  
  1. Drive and manage hardware
  2. Export better abstraction

- **OS is viewed as an extension of hardware**
  
  Privileged position – Only one OS

---

## Modern OS Evolution

### Goal – Support as many applications as possible

- **Problems** — Too complex
  
  - Security
  - Reliability
  - Manageability
  - Performance
  - Innovation

---

## Virtual Appliance Operating System as evolution beyond traditional OS model

- Don't need complex hardware management
- Don't need broad application support
- Application-specific operating system
- Smaller, easier to get right!
- Look at hardware appliance operating systems for examples

---

## Living in a Virtual World

- Virtualization makes OS flexible, portable, controllable and (potentially) fixable without internal changes
- Black boxes now manipulable objects
  
  OS, binary software, file systems, networks...
  
  Crack open, modify, ship around, monitor, etc.

- More opportunities for OS developers!
  
  Someone can run your custom OS (and keep Windows too)
  Application-specific OSes
  
  ... with less importance per opportunity
  
  OS doesn’t matter as much
  
  But now you know enough to hack VMMs as well as OSes!