# 6.033 Lecture 6 — OS Structure & Virtualization 2/24/2014

Last few lectures have focused on kernel design and OS features, in particular

- virtual memory, to protect two running processes from each other

- provides system calls so programs can interact, e.g.

    - send/receive (bounded buffer), a communication abstraction for programs running in different virtual memories
    - file system, etc

- threads, a way to virtualize the CPU so multiple processes can share it

Today:

- Can we rely on the OS kernel to work properly?

(Probably not 100% of the time)

- What to do about this?

(Introduce another layer of abstraction / virtualization!)

Structuring the kernel.
  [ slide: complexity of Unix v6, recent Linux ]
  Linux is effectively one large C program.

  Internally, no enforced modularity or client/server separation.

  Of course, lots of careful software engineering to make things work:

    Object-oriented programming style.

    Common interfaces between components.

    E.g., all storage devices implement the same set of functions,
      regardless of whether they're a USB drive, hard drive, CD, etc.

  However, one bug can cause entire system to malfunction or crash.
  This approach is often called a "monolithic" kernel.

Monolithic kernel: no enforced modularity (e.g., Linux).
  Many good software engineering techniques used in practice.
  But ultimately a bug in the kernel can affect entire system.

[ slide: kernel is complex ]
  We've seen this slide already, but this is critical to 6.033.
  One result of all this complexity: bugs!
  [ slide: 5000 bugs reported as fixed in ~7 years, or 2+ per day ]
  These bug counts are likely a fraction of actual # bugs fixed.

```
  guest# more main.c
    - small Linux kernel module: runs as part of the kernel.
    - every 10 seconds, randomly chooses N locations in physical memory.
    - writes a random byte to that location.
    - start with N=1, each time go up by a factor of 2x (1, 2, 4, 8, 16, ..)
```

```
   On madden's laptop running linux inside vmware:
   make a snapshot
   # less main.c
   # insmod scribble.ko
   # dmesg  (run periodically)
   ....
   restore snapshot
```

[slide observations]

Obviously not great that a kernel bug can cause a Linux system to fail.

  Is it a good thing that Linux lasted this long?

  Problems can be hard to detect, even though they may still do damage.
    E.g., maybe my files are now corrupted, but system didn't crash.

  Worse: adversary can exploit a bug to gain unauthorized access to system.
    Will look at security in much more detail towards end of 6.033.

**How can we deal with the complexity of such systems?**

Q:  could we design a operating system to be less prone to this problem?
A:  Yes — called a microkernel!

Enforces modularity by putting subsystems in "user" programs.
  E.g., file server, device driver, graphics, networking are just programs, that communicate with other programs using OS.

[slide: microkernel organization]

Why hasn't Linux been rewritten in a microkernel style?

Many dependencies between components in the kernel.
  E.g., balance between memory used for file cache vs process memory.

Moving large components to microkernel-style programs does not win much.
  If entire file system service crashes, might lose all data.

Better designs possible, but require a lot of careful design work.
  E.g., could create one file server program per file system,
    mount points tell FS client to talk to another FS server.

  Trade-off: spend a year on new features vs. microkernel rewrite.
  Microkernel design may make it more difficult to change interfaces later.

Some parts of Linux have microkernel design aspects:
  X server, some USB drivers, printing, SQL database, DNS, SSH, ...

    Can restart these components without crashing kernel or other programs.

**Problem: deal with Linux kernel bugs without redesigning Linux from scratch.**

One idea: run different programs on different computers — then they won't interfere with each other since each computer has its own Linux kernel; if one crashes, others not affected.

Strong isolation (all interactions are client-server), but often impractical.

Can't afford so many physical computers: hardware cost, power, space, ..

**Goal**: run multiple instances of Linux on a single computer. Will provide fault isolation, also allow us to have an OS environment customized for each program.

Sounds similar to running multiple programs on a single computer.

Original OS techniques: virtualization + abstractions.

New constraint: compatibility, because we want to run existing Linux kernel, without changing any of its code.

Linux kernel written to run on regular hardware.
So need to achieve this with no additional abstractions — just virtualization!

Virtual Machines

Approach is called "virtual machines" (VM): providing a virtual machine.
  Each virtual machine is often called a guest.

The equivalent of a kernel is called a "virtual machine monitor" (VMM).
  The VMM is often called the host.

[slide virtual machines]

My laptop was running Linux inside of a virtual machine for the demo.
  Provides enforced modularity.
  Even though Linux inside the VM failed, rest of system is fine.

Naive approach: can emulate every single instruction from the guest OS.
  In practice, this is sometimes done, but often too slow in practice.
[show slide]

Want to run most instructions directly on the CPU, like a regular OS kernel.

(Possible as long as the operating system binary we are trying to run uses the same instruction set as the host computer)

Computer state that must be virtualized for an existing OS kernel:
  Memory.
  Page table pointer (PTP), points to a page table in memory.
  U/K bit.
  More in practice (e.g., interrupts, registers), but techniques are similar.

[slide]  Diagram:

  OS kernel on top of HW, hardware contains memory, PTP, page table, U/K bit.
  Two copies of kernel + HW.
  VMM needs to implement virtual hardware using physical hardware.

<u>Virtualizing memory: at a high level idea as in an OS kernel.</u>
VMM must translate guest OS addresses into actual memory addresses
  E.g., if guest VM has 1GB of memory, it will access memory addrs 0..1GB, but VMM may map these to some other place in physical memory.

<u>Virtualizing the PTP register / page tables.</u>

*Can we load the address of guest VM's page table into PTP register?*
Terminology: 3 levels of addressing now.
  Guest virtual.
  Guest physical.
  Host physical.

$$\begin{tikzcd}
\text{Guest virtual address} \arrow[d, "\text{Kernel page table}"] \\
\text{Guest physical addresses} \arrow[d, "\text{VMM page table}"] \\
\text{Host physical addresses}
\end{tikzcd}$$

Guest VM's page table maps from guest virtual to guest physical addrs.

Hardware page table must point to host physical addrs (actual DRAM locations).

Setting hardware PTP register to point to guest page table would not work, since that would allow guest OS to choose which physical addresses it wants to access, which isn't what we want.

<u>One solution: shadow pages.</u>
[slides animation]
[ slide: set_ptp ]
VMM intercepts guest OS setting the virtual PTP register.
VMM iterates over the guest page table, constructs a corresponding shadow PT.
In shadow PT, every guest physical addr is translated into host physical addr.
Finally, VMM loads the host physical addr of the shadow PT.

<u>What happens if guest OS modifies its page table?</u>

Real hardware would start using the new page table's mappings.
(Some details omitted here..)

Virtual machine monitor has a separate shadow page table.

Goal: VMM needs to intercept when guest OS modifies page table,
update shadow page table accordingly.

Technique: use the read/write bit in the PTE to mark those pages read-only.

[slide — r/w bit]

If guest OS tries to modify them, hardware triggers page fault.
Page fault handled by VMM: update shadow page table & restart guest.

<u>Virtualizing the U/K bit.</u>

Hardware U/K bit should be set to 'U': otherwise guest OS can do anything

Behavior affected by the U/K bit:

1. Whether privileged instructions can be executed (e.g., load PTP).
2. Whether pages marked “read only” in page table can be modified.

*Implementing a virtual U/K bit:*
Approach is called "**trap and emulate**".

Guest OS runs in U mode

VMM stores the current state of guest's U/K bit (in some memory location).

Privileged instructions in guest code will cause an exception

VMM exception handler's job will be to emulate these instructions.

E.g., if load PTP instruction runs in virtual K mode, load shadow PT.

Otherwise, send an exception to the guest OS, so it can handle it.

(skip — but interesting side note)

How do we selectively allow / deny access to kernel-only pages in guest PT?
Hardware doesn't know about our virtual U/K bit.
Idea: generate two shadow page tables, one for U, one for K.
[ slide: set_ptp with kmode argument ]
When guest OS switches to U mode, VMM must invoke set_ptp(current, 0).

Hard: what if some instructions don't cause an exception?

E.g., on x86, can always read the current U/K bit value.
Guest OS kernel might observe it's running in U mode, expecting K.

Similarly, can jump to U mode from both U & K modes.
How to intercept switch to U mode to replace shadow page table?

Approach 1: "para-virtualization".

Modify the OS slightly to account for these differences.
Compromises on our goal of compatibility.
Recall: goal was to run multiple instances of the OS on a single computer.
  Not perfect virtualization, but relatively easy to change Linux.
  May be difficult to do with Windows, without Microsoft's help.

Approach 2: binary translation.

VMM analyzes code from the guest VM.
Replaces problematic instructions with an instruction that traps.
Once we can generate a trap, can emulate the desired behavior.
Original versions of VMware did this.

Approach 3: hardware support.

Recently (2005-08) AMD + Intel (VT-x) added virtualization support to their CPUs:
  Special "VMM" operating mode, in addition to U/K bit.
  Two levels of page tables, implemented in hardware.
Makes the VMM's job much easier:
  Let the guest VM manipulate the U/K bit, as long as VMM bit is cleared.
    VMM can choose which interrupts/exceptions it wants to handle
  Let the guest VM manipulate the guest PT, as long as host PT is set.
Many modern VMMs take this approach.

For more info see: http://web.mit.edu/6.033/2012/wwwdocs/papers/agesen.pdf

Virtualizing devices (e.g., disk), simplified:
OS kernel accesses disk by issuing special instructions to access I/O memory.

These instructions only accessible in K/VMM mode, and raise an exception.

VMM handles exception by simulating a disk operation.

Actual disk contents is typically stored in a file in the host file system.

“Trap and Emulate”, again!

Demo:
  What does a VM look like in a Linux system?
  E.g., a Linux virtual machine:

```
ls -lt "/Users/madden/Documents/Virtual Machines/Fedora 64-
bit.vmwarevm/"
```

Just a bunch of files on disk

  The entire VM is just another process on my Mac.

```
-bash ~ % ps -ef | grep vmware-vmx
    0 23412     1   0  8:44AM ??         2:18.53 /Applications/
VMware Fusion.app/Contents/Library/vmware-vmx -E en -D
5DWLxpu2G7OlhgEAAAAAAAAAAAAAAAAAAAAAAAAAAAA= -s
vmx.stdio.keep=TRUE -# product=128;name=VMware
Fusion;version=4.1.4;buildnumber=900582;licensename=VMware
Fusion for Mac OS;licenseversion=4.0+; -@ pipe=/var/folders/nw/
7p0zscrj0m90gj4jmb0dv3p40000gn/T//vmware-madden/
vmx1aa1e459f65bb70b;readyEvent=46 /Users/madden/Documents/
Virtual Machines/Fedora 64-bit.vmwarevm/Fedora 64-bit.vmx
  501 23690   709   0  9:29AM ttys000    0:00.00 grep vmware-vmx
```

Benefits of running an OS in a virtual machine. [slide]
  Can share hardware between unrelated services, with enforced modularity.
  Can run different OSes (Linux, Windows, etc).
  Level-of-indirection tricks: can move VM from one physical machine to another.

How do VMs relate to microkernels?
  Solving somewhat orthogonal problems.
  Microkernel: splitting up monolithic kernel into client/server components.
  VMs: how to run many instances of an existing OS kernel.
  E.g., can use VMM to run one network file server VM and one application VM.
  Hard problem still present.
    Entire file server fails at once?
    Or, more complex design with many file servers.

[ slide: summary ]