CIS 3207 - Operating Systems
CPU Mode

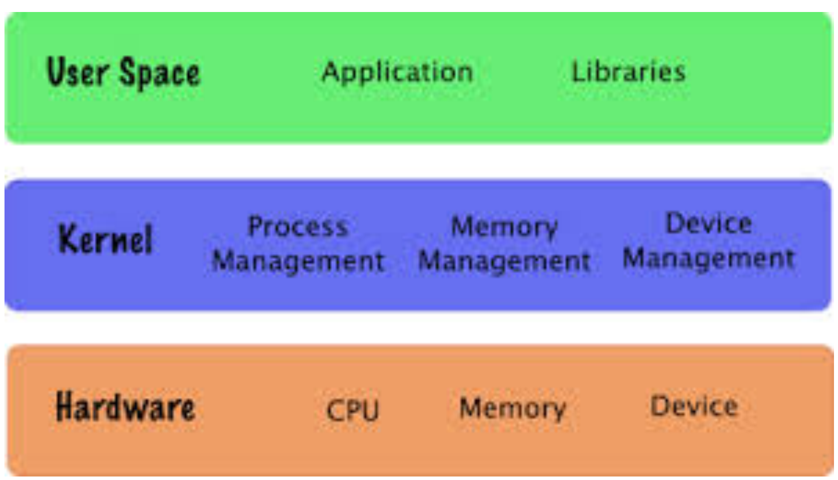
Professor Qiang Zeng

Spring 2018

TEMPLE UNIVERSITY

CPU Modes

- Two common modes
  - Kernel mode
    - The CPU has to be in this mode to execute the kernel code
  - User mode
    - The CPU has to be in this mode to execute the user code

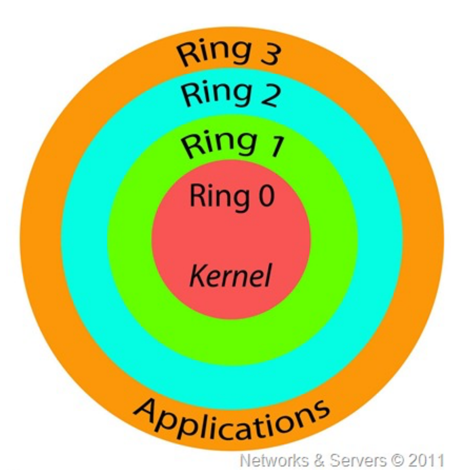


Important questions

- How are CPU modes implemented?
- Why are CPU modes needed?
- Difference between Kernel mode and User mode
- How are system calls implemented?
- Advanced topic: Virtualization

How CPU Modes are implemented

- Implemented through protection rings
  - A modern CPU typical provides different protection rings, which represent different *privilege levels*
    - A ring with a lower number has higher privileges
  - Introduced by Multics in 60's
  - E.g., an X86 CPU usually provides four rings, and a Linux/Unix/Windows OS uses Ring 0 for the kernel mode and Ring 3 for the user mode



Why are Protection Rings needed?

- Fault isolation: a fault (e.g., divided by 0) in the code running in a less-privileged ring can be captured and handled by code in a more-privileged ring
- Privileged instructions: certain instructions can only be issued in a privileged ring; thus an OS can implement resource management and isolation here
- Privileged memory space: certain memory can only be accessed in a privileged ring

All these are demonstrated in the difference between the kernel mode and the user mode

Kernel Mode vs. User Mode?

- A fault in the user space (e.g., *divided by zero*, *invalid access*, *null pointer dereference*) can be captured by the Kernel (without crashing the whole system)
  - Details of fault handling will be covered in later lectures
- Privileged instructions can only be issued in the kernel mode
  - E.g., disk I/O
  - In X86, an attempt to execute them from ring 3 leads to GP (General Protection) exceptions
- The kernel memory space can only be accessed in the kernel mode
  - E.g., the list of processes for scheduling

What are the “real mode” “protected mode” in x86 CPUs

In “real mode”, protection rings are NOT enforced, while in “protected mode”, **protection rings** are enforced

Examples of Privileged Instructions

- I/O operations
- Switch page tables of processes: load cr3
- Enable/disable interrupts: sti/cli
- Change processor modes from kernel to user: iret
- Halt a processor to enter low-power stage: hlt
- Load the *Global Descriptor Table* register in x86: lgdt

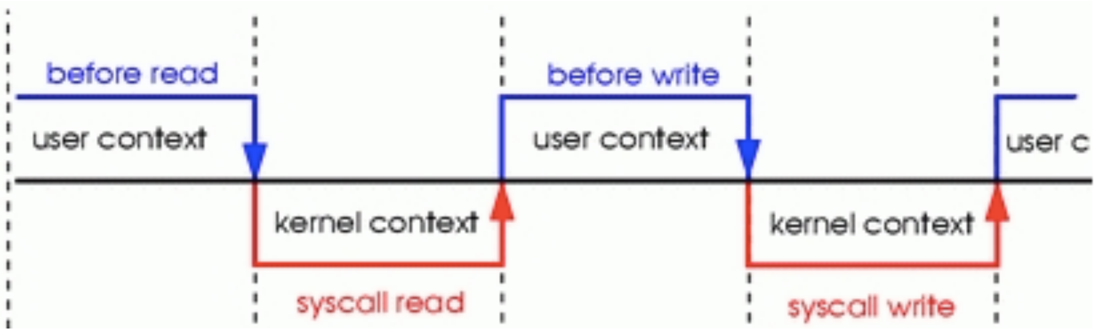
- Ref: http://www.brokenthorn.com/Resources/OSDev23.html

- Examples of non-privileged ones:
  - add, sub, or, etc.

Questions

- If I/O operations rely on privileged instructions, how does a user program read/write?
  - System calls
  - When a system call is issued, the process goes from user mode (Ring 3) to kernel mode (Ring 0)
  - `printf` libc call (Ring 3) => `write` system call => Kernel code (Ring 0)

A CPU enters user mode and kernel mode in an interleaved way



How to interpret the output of the `time` command

```
$ time any-command
        real    0m1.734s
        user    0m0.017s
        sys     0m0.040s
```

- Real: wall clock time
- User: CPU time spent in user-mode
- Sys: CPU time spent in kernel-mode
- Actual CPU time: user + sys
- Why “real != user + sys”?

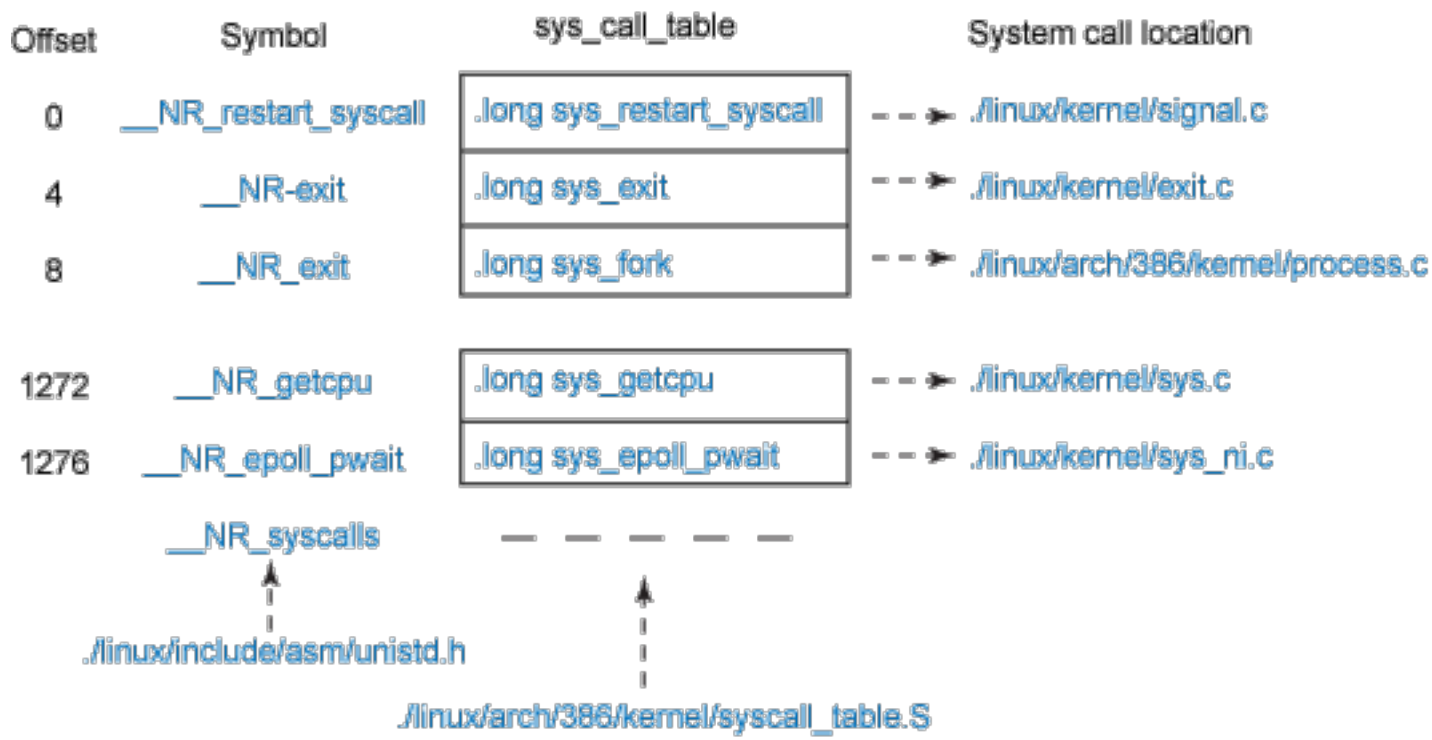
Myth: "root" refers to the kernel mode?

- Short answer: no!
- Long answer: the root user and non-root user refer to the user account types; in Linux/Unix, the root user can access any files, while a non-root user only has access to some files.
- Kernel Mode and User Mode refer to the processor mode
- No matter the user is a root or non-root, a CPU still enter Kernel mode and User mode in an interleaved way
- Regardless of the current CPU mode, a root user is always a root user
- That is, they are orthogonal concepts

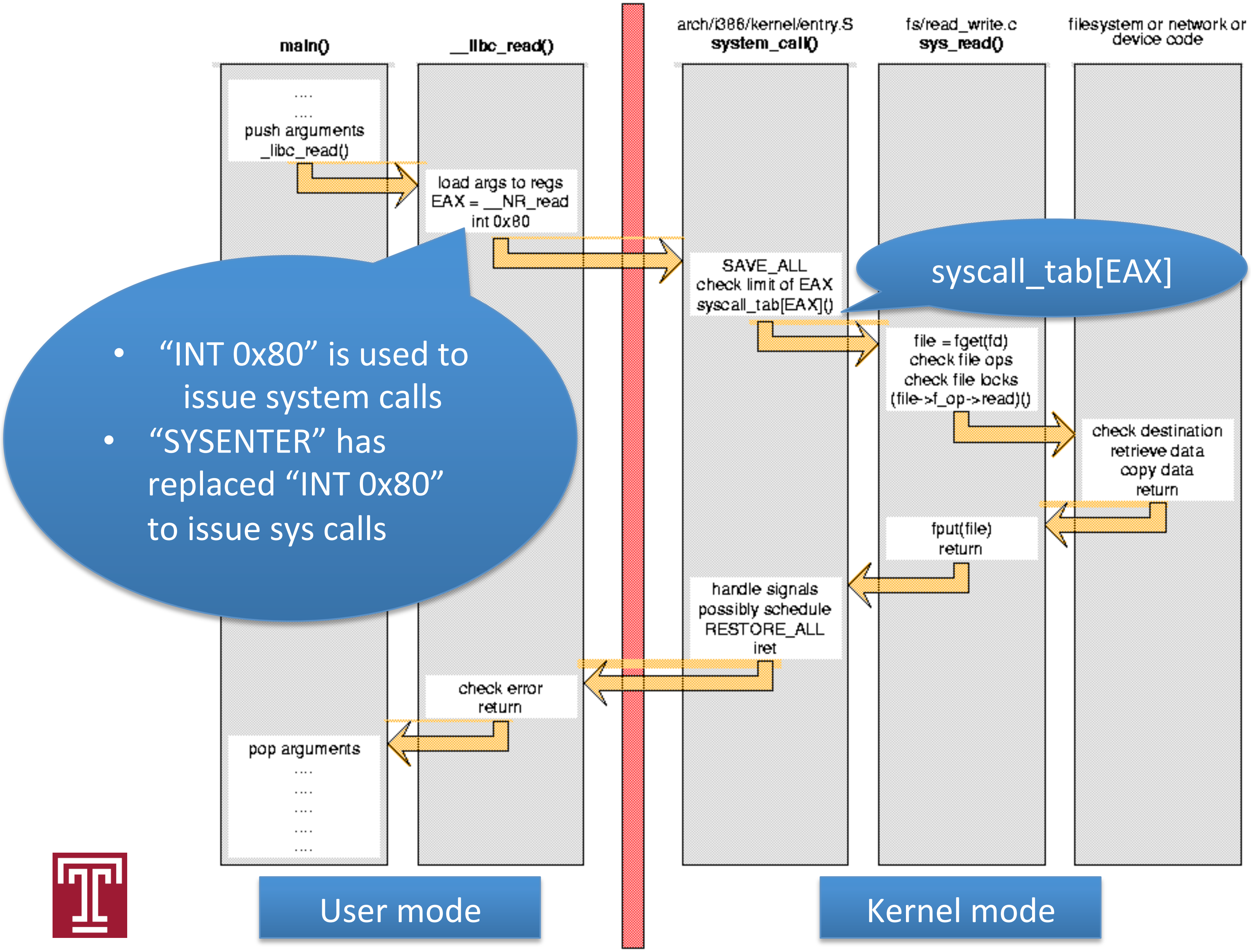
How will you design the mechanism of System Calls?

- Given a system call, how to design the CPU and the kernel to execute it?
- **Background**: the Program Counter (PC) register in a processor stores the address of the instruction to be executed
  - PC is incremented after fetching an instruction
  - But "jump", "call" and "ret" instruction can set the PC value
- If the user code can set the PC register *arbitrarily* before changing from Ring 3 to Ring 0, how will you exploit the kernel code?
  - This is very dangerous, as the user code can exploit the power of Ring 0 to harm the whole system

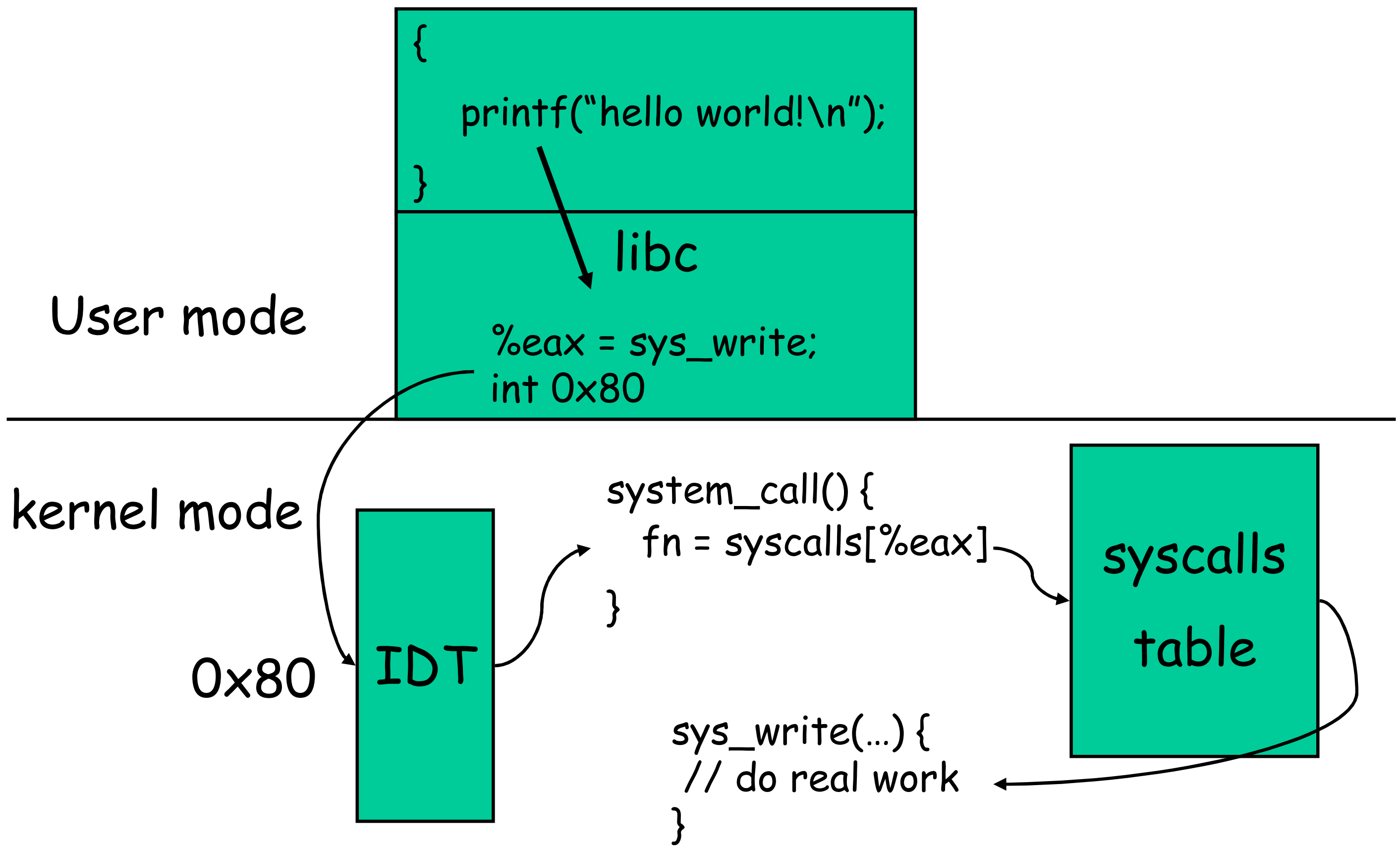
System Call Table



System calls in Linux



Linux system call overview



Graph by Dr. Junfeng Yang

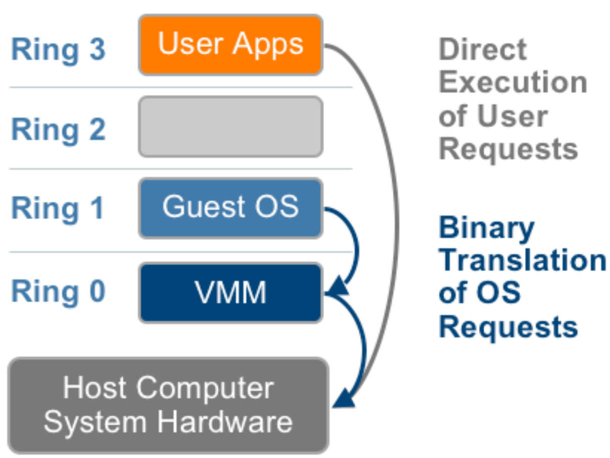
How to trace system calls in Linux/Unix

- “strace” command can trace system calls
- “ltrace” command can trace library calls

```
qiang@ubuntu:~$ strace ls
execve("/bin/ls", ["ls"], [/* 65 vars */]) = 0
brk(0)                                  = 0x944000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fbfa8456000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=96716, ...}) = 0
mmap(NULL, 96716, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fbfa843e000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0[\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=134296, ...}) = 0
mmap(NULL, 2238192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fbfa8013000
mprotect(0x7fbfa8033000, 2093056, PROT_NONE) = 0
mmap(0x7fbfa8232000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1f000) = 0x7fbfa823200
0
mmap(0x7fbfa8234000, 5872, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fbfa8234000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libacl.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0`\34\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=31168, ...}) = 0
mmap(NULL, 2126336, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fbfa7e0b000
mprotect(0x7fbfa7e12000, 2093056, PROT_NONE) = 0
mmap(0x7fbfa8011000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x7fbfa8011000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
```

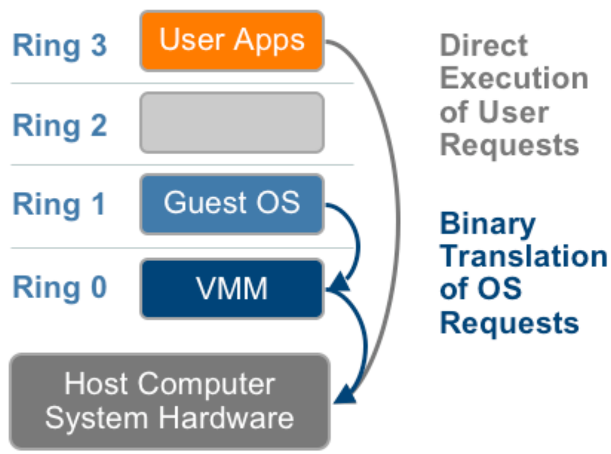
Rings and Virtualization

- A Hypervisor is a Virtual Machine Monitor (VMM) that runs and manages virtual machines
- A straightforward virtualization scheme
  - Hypervisor: Ring 0; VM Kernel: Ring 1; VM User: Ring 3
  - But there are instructions in X86 (sensitive but non-privileged) that cause problems when running in Ring 1; e.g., *SGDT* returns the host GDT info
  - The hypervisor is supposed to handle them
    - E.g., the hypervisor can maintain a virtual GDT for each VM and returns the VM's GDT info when SGDT is invoked from a VM



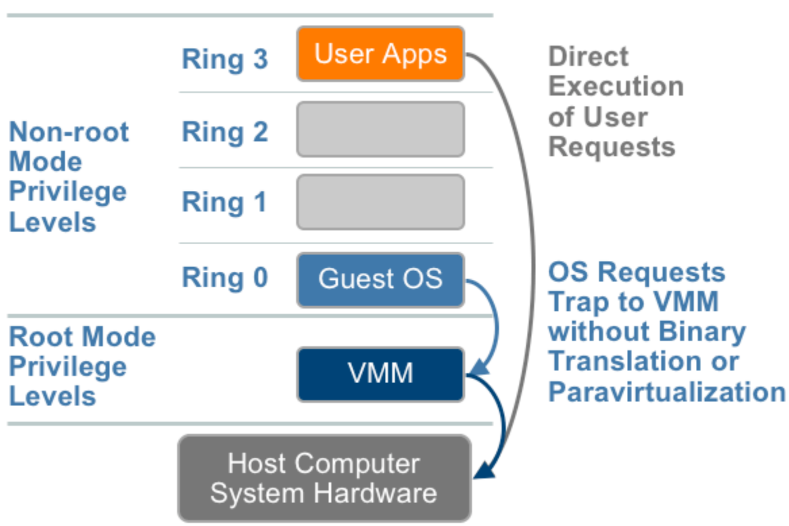
Rings and Virtualization

- How to handle those instructions?
  - Binary translation (e.g., full-virtualization in certain **VMware versions**)
  - Modification of the guest OS (e.g., via para-virtualization in certain **Xen versions**)



Ring -1 used by the Hypervisor

- In 2005 and 2006, Intel and AMD introduced Ring -1, respectively; it is used by the Hypervisor
  - The VM kernel uses Ring 0, and the Hypervisor -1
  - The Hypervisor can configure with the CPU which instructions are of interest, so whenever they are executed, the execution traps from Ring 0 to -1
  - Hardwar-assisted full virtualization



Take-away

- CPU provides protection rings, while an OS use them for the kernel mode and the user mode
- A fault in the user code will not crash the system
- User code cannot do I/O directly, but do it through system calls
- The design of system calls is beautiful, because..
  - they allow your program to do something powerful; in the meanwhile you cannot abuse them easily
- Ring -1 is used by the Hypervisor

What else?

Three very useful Linux/Unix commands:

time
strace
ltrace

Required Readings

- Rings
  - http://duartes.org/gustavo/blog/post/cpu-rings-privilege-and-protection/
- System calls
  - https://www.ibm.com/developerworks/linux/library/l-system-calls/
  - https://www.cs.columbia.edu/~junfeng/10sp-w4118/lectures/l05-syscall-intr-linux.pdf
  - https://www.cs.princeton.edu/courses/archive/fall10/cos318/lectures/OSStructure.pdf
  - https://elixir.free-electrons.com/linux/v2.6.18-rc6/source/arch/i386/kernel/vsyscall-sysenter.S
  - Protecting Against Unexpected System Calls. Usenix Sec'05.
  - System call issues: http://www.inf.ed.ac.uk/teaching/courses/os/slides/02-operations.pdf
- Compatibility
  - http://rlc.vlinder.ca/blog/2009/08/binary-compatibility/

Optional Readings on Virtualization

- Introduction of Xen
  - "Xen and the art of virtualization." SOSP '03
  - Slides for the paper above: http://courses.cs.vt.edu/cs5204/fall14-butt/lectures/xen.pdf
  - Virtualization in Xen 3.0. Rami Rosen, Linux Journal 2006.
  - "The definitive guide to the xen hypervisor." 2008
- Introduction of Vmware
  - Virtual Machines & VMware, Part I by Jay Munro
  - VMware and CPU Virtualization Technology
  - Virtualization-optimized architectures
  - Marshall, David. "Understanding Full Virtualization, Paravirtualization, and Hardware Assist."
  - Error in the paper above
- Very good slides
  - https://cbw.sh/static/class/5600/slides/11_Virtual_Machines.pptx
  - https://www.ics.uci.edu/~aburtsev/cs5460/lectures/lecture25-virtualization/lecture25-virtualization.pdf
- List of hypercalls
  - https://xenbits.xen.org/docs/unstable/hypercall/x86_64/include,public,xen.h.html
- Survey
  - Hwang, Jinho, et al. "A component-based performance comparison of four hypervisors." 2013.
  - A summary of virtualization techniques, Haro et al. 2012
  - **A Comparison of Software and Hardware Techniques for x86 Virtualization, ASPLOS'06**
  - **Virtualization Basics: Understanding Techniques and Fundamentals, Lee et al., 2014**

Sensitive but non-privileged instructions

- Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor. Usenix Security, 2000.
- Intel Privileged and Sensitive Instructions
- https://stackoverflow.com/questions/32794361/what-are-non-virtualizable-instructions-in-x86-architecture

Optional Readings on x86-64

- Long mode = 64-bit mode + Compatibility mode
  - https://en.wikipedia.org/wiki/Long_mode
- Legacy mode: 64-bit programs cannot run
  - https://en.wikipedia.org/wiki/X86-64
- Real mode (without protection rings) vs. V86 mode (virtual real mode in protected mode) vs. Protected Mode
  - https://stackoverflow.com/questions/43111970/whats-the-difference-between-virtual-8086-mode-and-real-address-mode-in-x86-pro
- Why do 32-bit applications work on x86-64 CPU?
  - https://stackoverflow.com/questions/28307180/why-do-32-bit-applications-work-on-64-bit-x86-cpus
- How retiring segmentation in AMD64 long mode broke Vmware
  - http://www.pagetable.com/?p=25

Ring -2: System Management Mode

- https://security.stackexchange.com/questions/129098/what-is-protection-ring-1

CPL, DPL, RPL (RPL not used nowadays)

- DPL - Descriptor Privilege Level
- CPL - Current Privilege Level
  - The CPL bits are always consistent with the kernel/user mode
- RPL - Requested Privilege Level
- A logical addr = a 16-bit segment identifier/selector + offset
  - 13-bit index + 1-bit Table indicator + 2-bit RPL
- The Intel processor provides Segment Registers to hold Segment Selectors
  - cs (it contains CPL), ss, ds
- A Segment Descriptor has 8 bytes and contains DPL
- Before the processor loads a segment selector into a segment register, it performs a privilege check:
  - Max(CPL, RPL) <= DPL
  - https://software.intel.com/en-us/forums/intel-isa-extensions/topic/733754
  - https://iambvk.wordpress.com/2007/10/10/notes-on-cpl-dpl-and-rpl-terms/