Slides last updated:
22.09.2025

Computer Systems

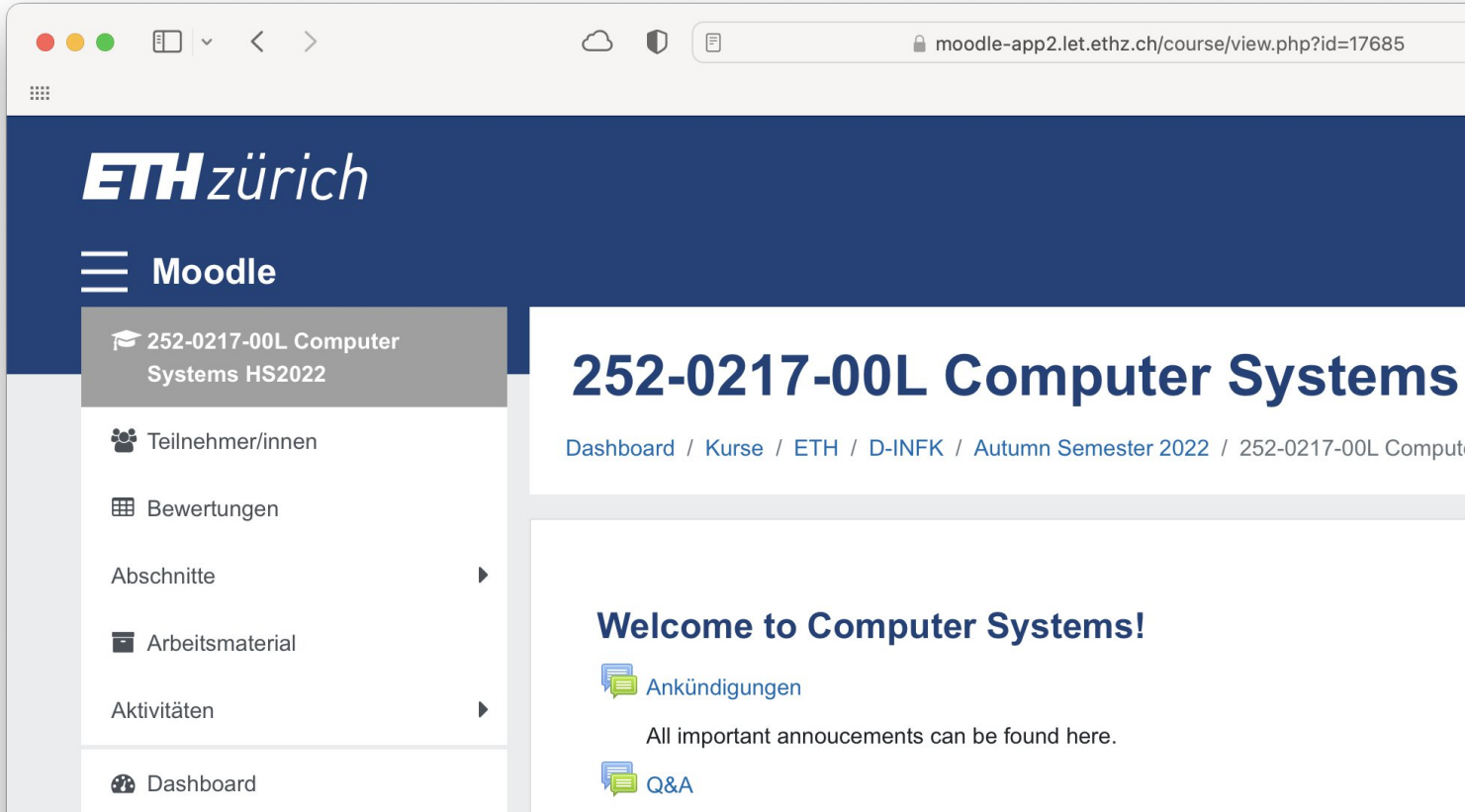
Exercise Session 1
HS 2026

Program

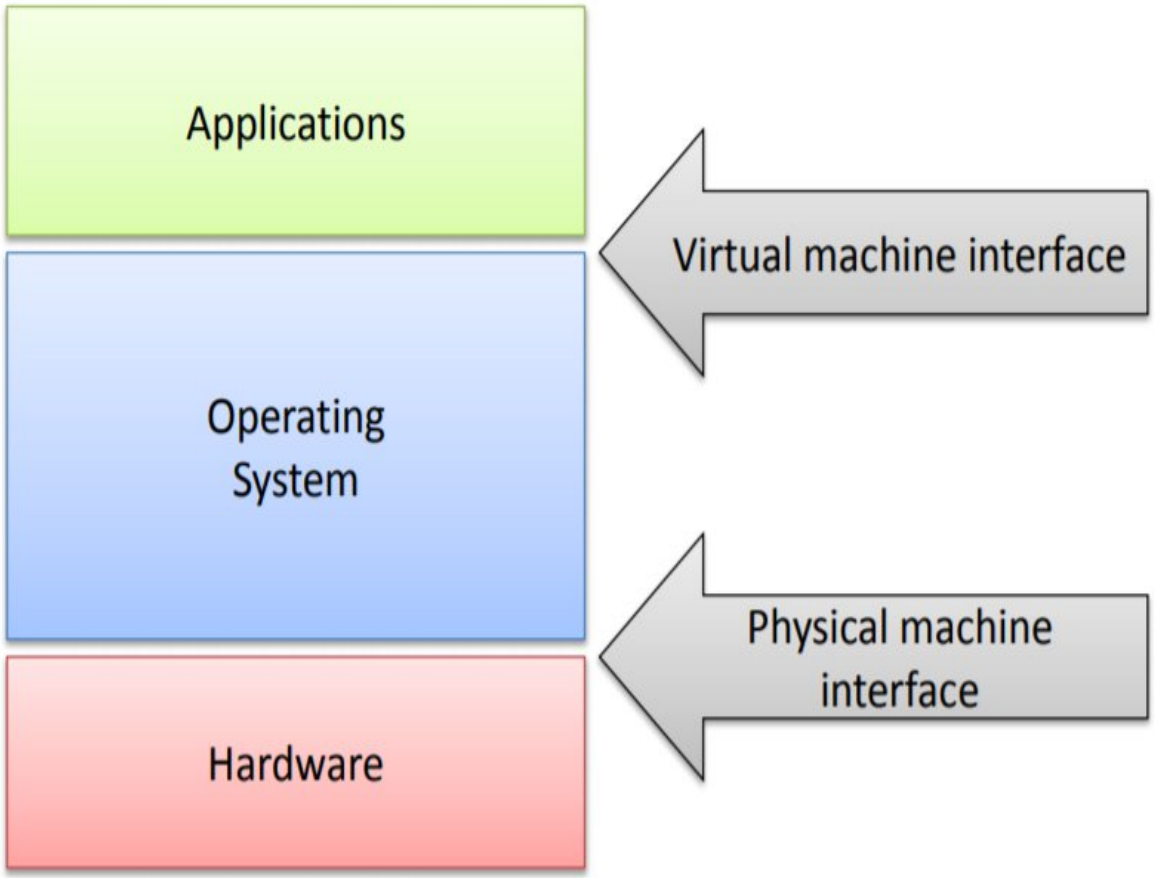
1. $whoami
2. Logistics
3. Quiz
4. Assignment preview

Logistics I

- Course Website: https://moodle-app2.let.ethz.ch/course/view.php?id=29335
- All course material will be uploaded to Moodle
- Use the Q & A forum to ask questions



The Role of the OS



Referee:

- Ensure resource sharing
- Ensure protection (memory protection, process isolation)
- Ensure Inter-process communication

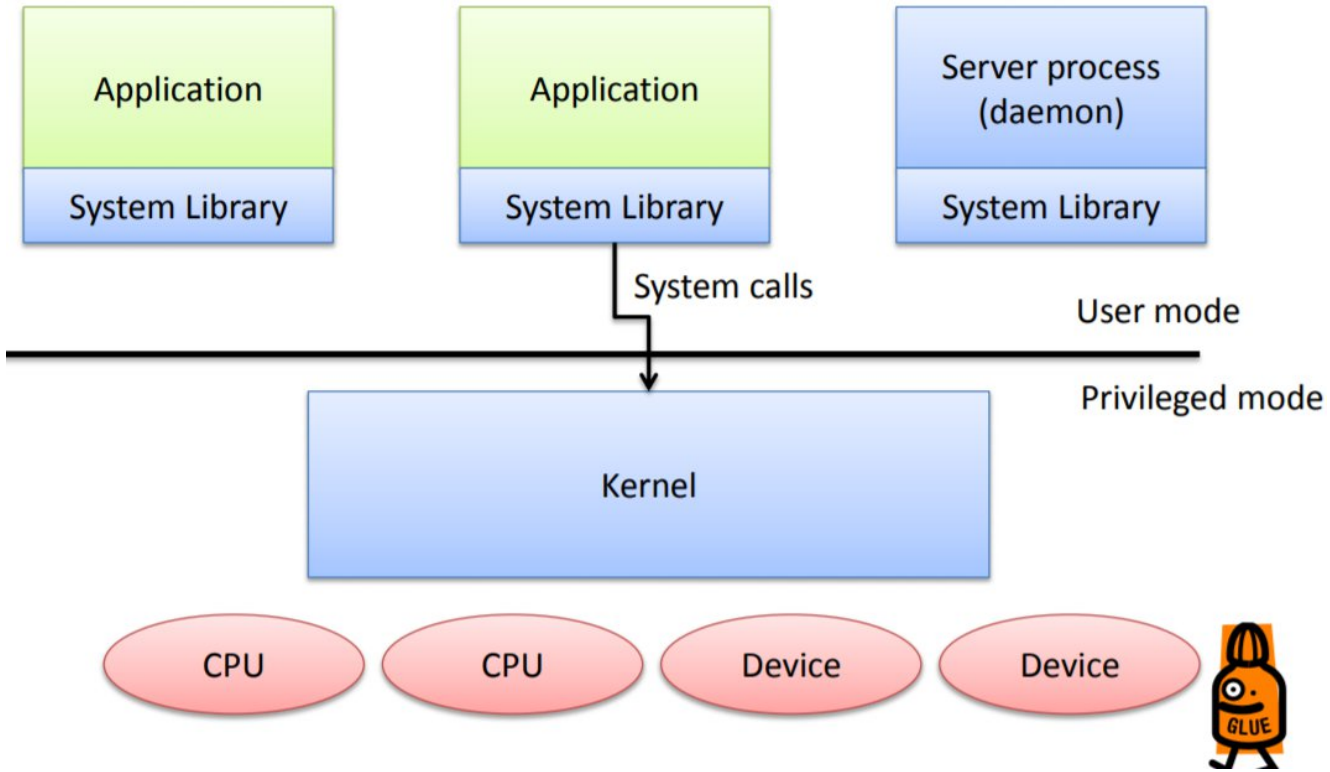
Illusionist:

- provide virtual resources to user-space processes
- Virtual Memory (full address space)
- Shared network interface

Glue:

- Provide high-level abstraction to user-space applications

General OS Structure



Kernel:

- Special process that runs in privileged mode
- Typically event driven server

System Library:

- Convenience functions (strcmp() etc.)
- System call wrappers

Daemon:

- Processes which are part of the OS but live in user-space
- Provides modularity, fault-tolerance

Daemons ?

Systems@ETH zürich

systemd(1) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | UNITS | DIRECTORIES | SIGNALS | ENVIRONMENT | KERNEL COMMAND LINE | SYSTEM CREDENTIALS | READINESS PROTOCOL | OPTIONS | SYSTEM CLOCK EPOCH | FILES | HISTORY | SEE ALSO | NOTES | COLOPHON

Search online pages

***SYSTEMD*(1)** **systemd** ***SYSTEMD*(1)**

NAME top

systemd, init - systemd system and service manager

SYNOPSIS top

/usr/lib/systemd/systemd [OPTIONS...]

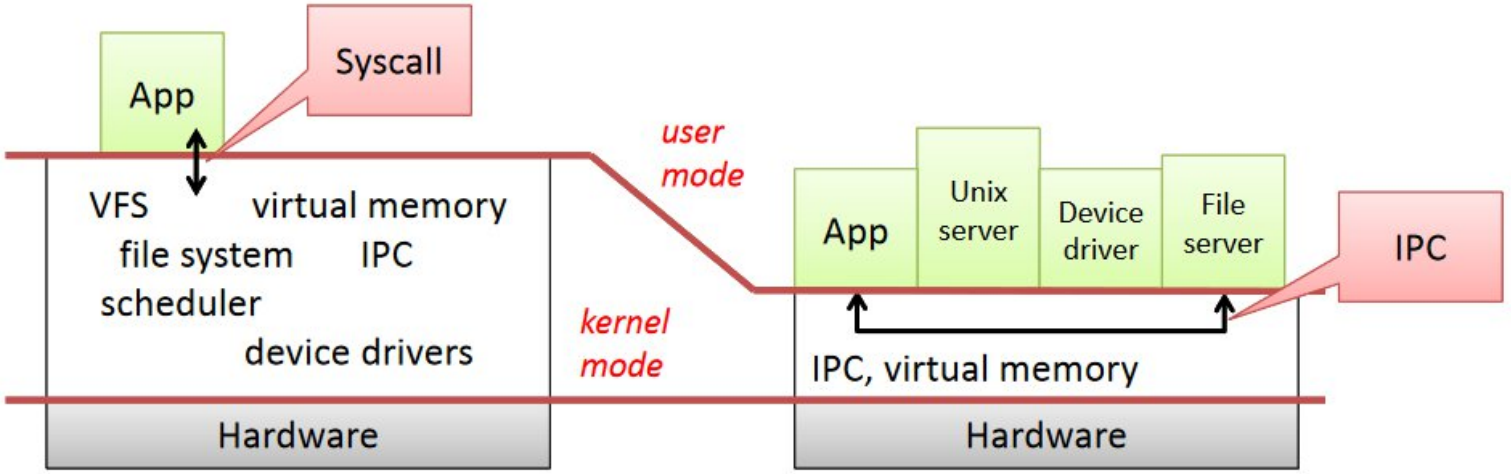
init [OPTIONS...]

DESCRIPTION top

systemd is a system and service manager for Linux operating systems. When run as first process on boot (as PID 1), it acts as init system that brings up and maintains userspace services. Separate instances are started for logged-in users to start their services.

systemd is usually not invoked directly by the user, but is installed as the /sbin/init symlink and started during early boot. The user manager instances are started automatically through the user@.service(5) service.

Monolithic vs. Microkernel



- Monolithic OS
  - lots of privileged code
  - services invoked by syscall
- Microkernel OS:
  - little privileged code
  - services invoked by IPC
  - “horizontal” structure

Exokernel: move functionality into system libraries instead of user-space

Multikernel: run different kernels on different cores (e.g. Barrelfish)

Bootstrapping

1. Power on
2. Load BIOS from ROM to memory
3. BIOS loads boot loader into mem
4. boot loader loads OS (kernel) fro
5. Transfer control to OS

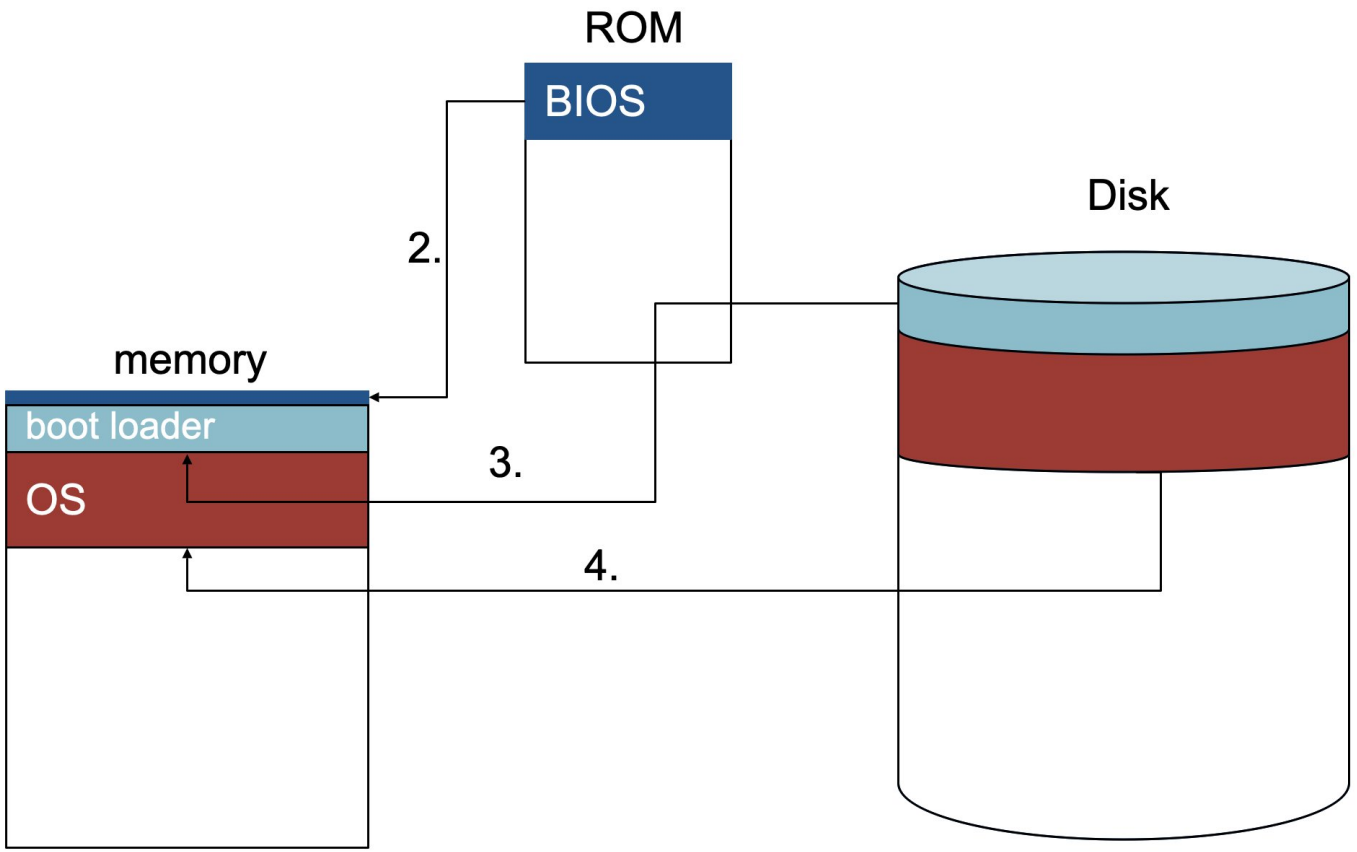
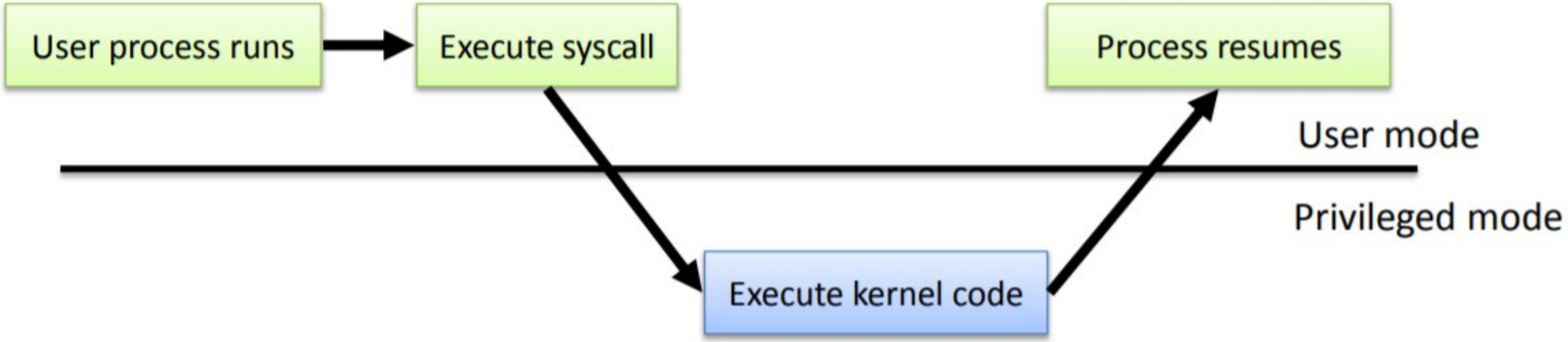


Illustration adapted from Maegan Pitman

Entering and Leaving the Kernel

- On Start-Up
- Exception occurs (caused by program)
- Interrupt occurs (caused by “something” else)
- upon a System Call



System Calls

- **All communication between user space and the kernel**
  - Reading, writing to disk, network, IPC, ...
  - Find out what your program is doing: *strace <program> / strace –p <PID>*

1.) **“int 80”** —> Software generated interrupt (syscall number in register)

2.) processor switches into privileged mode and jumps to”interrupt_vec[80]”

(depending on arch, processor does more or less (eg. Automatically switching stack pointer etc..))

3.) system call handler is executed

4.) Return —> Change back to “unprivileged mode” and continue in user-space

System Calls in xv6 – More detailed

https://pdos.csail.mit.edu/6.1810/2024/xv6/book-riscv-rev4.pdf (Chapter 4 – Traps and systems calls)

How does GDB set breakpoints, and then allow you to inspect the registers of the process ?

1. GDB informs the kernel that we want to trace that process
2. Write an invalid instruction at the start of the function we want to set a breakpoint on
3. Tracee calls invalid instruction ==> exception ==> kernel ==> kernel sees that GDB wants to inspect this ==> notifies GDB
4. GDB can request contents of registers that are now inside the trapframe

Additional Resources

xv6 Book

https://pdos.csail.mit.edu/6.1810/2024/xv6/book-riscv-rev4.pdf

Xv6 Exercises / Labs

https://pdos.csail.mit.edu/6.1810/2024/xv6.html

More info about Linux Kernel

https://kernel-internals.org

Quiz

What is the difference between a **virtualization** and a **transparency**?

Quiz

What is the difference between a **virtualization** and a **transparency**?

- Virtualization: a system appears to be a different system, or a larger system, or many individual systems, …

Quiz

What is the difference between a **virtualization** and a **transparency**?

- Virtualization: a system appears to be a different system, or a larger system, or many individual systems, …
- Transparency: a system hides some or all of its characteristics – it cannot be observed

Quiz

What is the difference between a **virtualization** and a **transparency**?

- Virtualization: a system appears to be a different system, or a larger system, or many individual systems, …
- Transparency: a system hides some or all of its characteristics – it cannot be observed

How does a Hypervisor know that it is not being virtualized?

Quiz

What is the difference between a **virtualization** and a **transparency**?

- Virtualization: a system appears to be a different system, or a larger system, or many individual systems, …
- Transparency: a system hides some or all of its characteristics – it cannot be observed

How does a Hypervisor know that it is not being virtualized?

- Unknowable. Individual instructions may be emulated.

Quiz

What is the difference between a **virtualization** and a **transparency**?

- Virtualization: a system appears to be a different system, or a larger system, or many individual systems, …
- Transparency: a system hides some or all of its characteristics – it cannot be observed

How does a Hypervisor know that it is not being virtualized?

- Unknowable. Individual instructions may be emulated.

What is the advantage of running a process in **userspace**?

Quiz

What is the difference between a **virtualization** and a **transparency**?

- Virtualization: a system appears to be a different system, or a larger system, or many individual systems, …
- Transparency: a system hides some or all of its characteristics – it cannot be observed

How does a Hypervisor know that it is not being virtualized?

- Unknowable. Individual instructions may be emulated.

What is the advantage of running a process in **userspace**?

- A failing process is easier to handle in userspace.

Quiz

What is the difference between a **virtualization** and a **transparency**?

- Virtualization: a system appears to be a different system, or a larger system, or many individual systems, …
- Transparency: a system hides some or all of its characteristics – it cannot be observed

How does a Hypervisor know that it is not being virtualized?

- Unknowable. Individual instructions may be emulated.

What is the advantage of running a process in **userspace**?

- A failing process is easier to handle in userspace.

Where, could a kernel exception handler save user program registers?

Quiz

What is the difference between a **virtualization** and a **transparency**?

- Virtualization: a system appears to be a different system, or a larger system, or many individual systems, …
- Transparency: a system hides some or all of its characteristics – it cannot be observed

How does a Hypervisor know that it is not being virtualized?

- Unknowable. Individual instructions may be emulated.

What is the advantage of running a process in **userspace**?

- A failing process is easier to handle in userspace.

Where, could a kernel exception handler save user program registers?

Quiz

What is the difference between a **virtualization** and a **transparency**?

- Virtualization: a system appears to be a different system, or a larger system, or many individual systems, …
- Transparency: a system hides some or all of its characteristics – it cannot be observed

How does a Hypervisor know that it is not being virtualized?

- Unknowable. Individual instructions may be emulated.

What is the advantage of running a process in **userspace**?

- A failing process is easier to handle in userspace.

Why, and where, could a kernel exception handler save user program registers?

- Where: Per-process trapframe or exception-specific trapframe

Quiz (continued)

Algorithm 3.15 System call for `write( fd, buf, count )`

Procedure in user space

1: Load `fd`, `buf`, and `count` into processor registers
2: Load system call number for `write` into a register
3: Trap
4: Read result from a register
5: **return** Result

Execution in the kernel (the trap handler)

6: Set up execution environent (stack, etc.)
7: Read system call number from register
8: Jump to `write` code based on this
9: Read `fd`, `buf`, and `count` from processor registers
10: Check `buf` and `count` for validity
11: Copy `count` bytes from user memory into kernel buffer
12: Do the rest of the code for `write`
13: Load the return value into a register
14: Resume the calling process, transfer to user mode

What is the order in which the individual steps happen?

Quiz (continued)

Algorithm 3.15 System call for `write( fd, buf, count )`

Procedure in user space

1: Load `fd`, `buf`, and `count` into processor registers
2: Load system call number for `write` into a register
3: Trap
4: Read result from a register
5: **return** Result

Execution in the kernel (the trap handler)

6: Set up execution environent (stack, etc.)
7: Read system call number from register
8: Jump to `write` code based on this
9: Read `fd`, `buf`, and `count` from processor registers
10: Check `buf` and `count` for validity
11: Copy `count` bytes from user memory into kernel buffer
12: Do the rest of the code for `write`
13: Load the return value into a register
14: Resume the calling process, transfer to user mode

What is the order in which the individual steps happen?

1, 2, 3, 6 - 14, 4, 5

Assignment 1 Overview

- Compile (and modify) the linux kernel
- Build the smallest bootable kernel
- Compile xv6
- Optional: Compile the Barrelfish OS
- Read the instructions carefully!!