DS and OS Research

CS503: Operating systems, Spring 2019

Pedro Fonseca
Department of Computer Science
Purdue University

Admin

- Next week:
  - Tuesday (4/23): exam review
    - No quiz
  - Thursday (4/25): no class due to evening midterm
- Exam: **Thu 05/02, 8:00a - 10:00a @ FRNY B124**

Previous lecture

- Linux:
  - Clone system call
  - Sys file system
  - Virtual memory
  - Buddy allocator
  - Slab allocator
  - BPF
- Micro-kernel vs. monolithic kernel

What is a distributed system?

- “A collection of independent, autonomous hosts connected through a communication network.”
- “A distributed system is a collection of services accessed via network-based interfaces”
- “Collection of independent computers that appears as a single system to the user(s)”
- “A group of computers working together as to appear as a single computer to the end-user.”
- “A distributed system is a system that prevents you from doing any work when a computer you have never heard about, fails.”

What is a distributed system?

- Independent computation units
- …that communicate between each other…
- …and somehow create the appearance of a single unit

Fundamental differences between OS and DS environments

- No shared memory
- No shared clock

Distributed systems challenges

- Latency of communication
- Fault-tolerance
    - Dropped messages, crashed nodes
- Concurrency:
    - Code running on different nodes
- Large-scale systems
- Reason about the properties of the system

Properties

- Reasoning about the correctness of systems
  - Think about all possible execution traces
  - Check that correctness properties hold for them
- Correctness properties:
  - **Safety** properties: asserts that nothing bad happens
  - **Liveness** properties: asserts that something good eventually

Discussion topics

- Are messages in OS settings always reliable
- Differences between concurrency bugs and distributed system bugs
- Multi-cores vs. “multi-machines”

Distributed system meet operating systems

- Distributed shared memory
- Distributed operating systems

Distributed shared memory

- What is distributed shared memory?
- How to implement distributed shared memory?
- What are the challenges?

Distributed operating system

- Q: What would distributed operating system mean?
- Applications running on different machines:
  - Invoke system calls and communicate as if they were on the same machine
- Applications running across machines:
  - One thread on each machine and migration of machines

Distributed operating system

- A single global IPC mechanism (any process should be able to talk to any other process in the same manner, whether it's local or remote).
- A global protection scheme.
- Uniform naming from anywhere; the file system should look the same.
- Same system call interface everywhere.

Quick note

- For this course, assume that exam questions are not about distributed system settings unless otherwise stated

Break

OS research

Systems research (focus on OS)

- SOSP / OSDI
- EuroSys
- ATC
- ASPLOS (+ architecture)
- PLDI (+ PL)

Systems research

- Most of the research falls into:
  - More efficient systems
  - More reliable and secure systems
  - Easier to program
- Some of the research is driven by hardware changes or “paradigm shifts”
- Not all system research involves low-level code

Some classical papers

- The Working Set Model for Program Behavior. CACM 1968.
- The UNIX Time-Sharing System. CACM 1974.
- Formal Requirements for Virtualizable Third Generation Architectures. CACM 1974.
- Experiences with Processes and Monitors in Mesa. CACM 1980.
- Scheduling Techniques for Concurrent Systems. ICDCS 1982.
- An Implementation of a Log-Structured File System for UNIX. ATC 1993.
- Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions. OSDI 2000.

Recent papers at: https://jeffhuang.com/best_paper_awards.htm

MultiNyx: A Multi-Level Abstraction Framework for Systematic Analysis of Hypervisors

Pedro Fonseca, Xi Wang, Arvind Krishnamurthy

EuroSys'18

What is the problem addressed?

- Hypervisors need to virtualize correctly all the architecture details
  - 1000s of pages describing architecture
  - Developers need to understand and virtualize correctly
- Hypervisor bugs cause applications to crash, information leakage, etc.
- Modern hypervisors are implemented with **CPU virtualization extensions**

What is the problem addressed?

Bug example: KVM bug (CVE-2017-2583)

- Incorrect MOV virtualization:
  - VM crash (Intel)
  - Privilege escalation (AMD)
- Several conditions are required to trigger the bug:
  - MOV has to be emulated by the VMM
  - MOV has to load a NULL stack segment
  - Had to be executed in long mode and with CPL=3
  - Other privilege related fields had to have specific values (SS.RPL=3, SS.DPL=3)

How to generate effective tests for modern hypervisors?

Which are the previous approaches?

- Manual tests
- Stress testing
- Black-box testing
  - Create test cases based on spec

What is our approach?

- 1. Systematically generate test cases using **symbolic execution (white-box testing)**
  - Challenge 1: How to model complex instructions?
  - Challenge 2: How to make symbolic execution scale?
- 2. Analyze test cases through **differential testing**

Challenge 1:
How to model complex instructions?

Hypervisors use complex, system-level instructions

```
function hypervisor(input){
  x = VMENTER(input);
  if(x == 1000){
    assert(0); // Crash
  }
  return;
}
```

Virtualization instruction

How to encode complex instructions?

Phase 1:
Encode constraints

Hypervisor → `??? == 1000`

MultiNyx approach

```
function hypervisor(input){
  x = VMENTER(input);
  if(x == 1000){
    assert(0); // Crash
  }
  return;
}
```

CPU Simulator

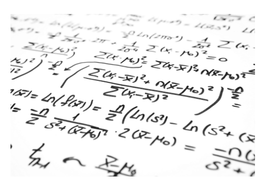
```
function VMENTER(input){
  x = input;
  x = x + 4
  …
  return x;
}
```

Phase 1:
Encode constraints

Phase 2:
Solve constraints

Hypervisor

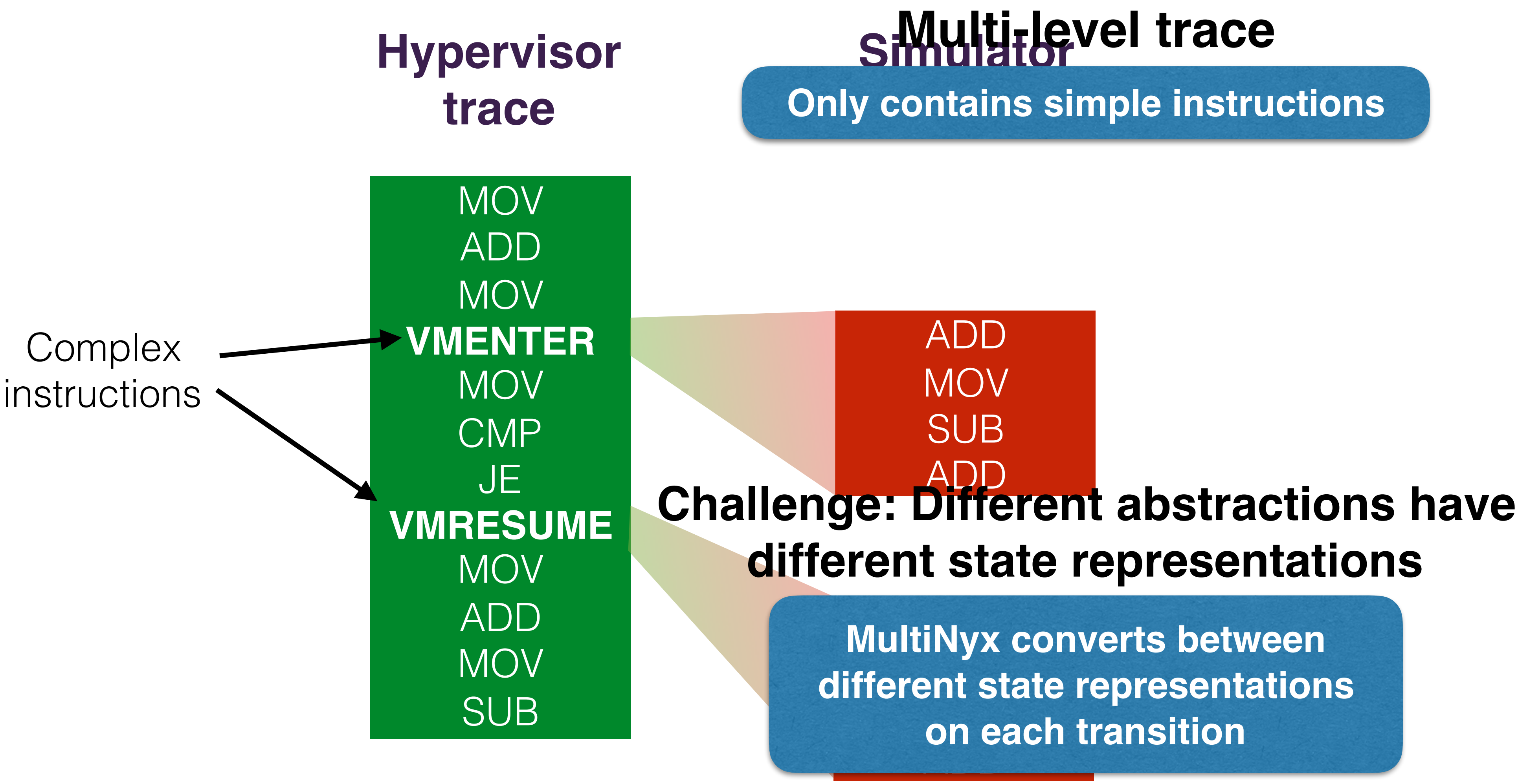
CPU simulator



Path constraints

Test cases

MultiNyx: multi-level symbolic execution



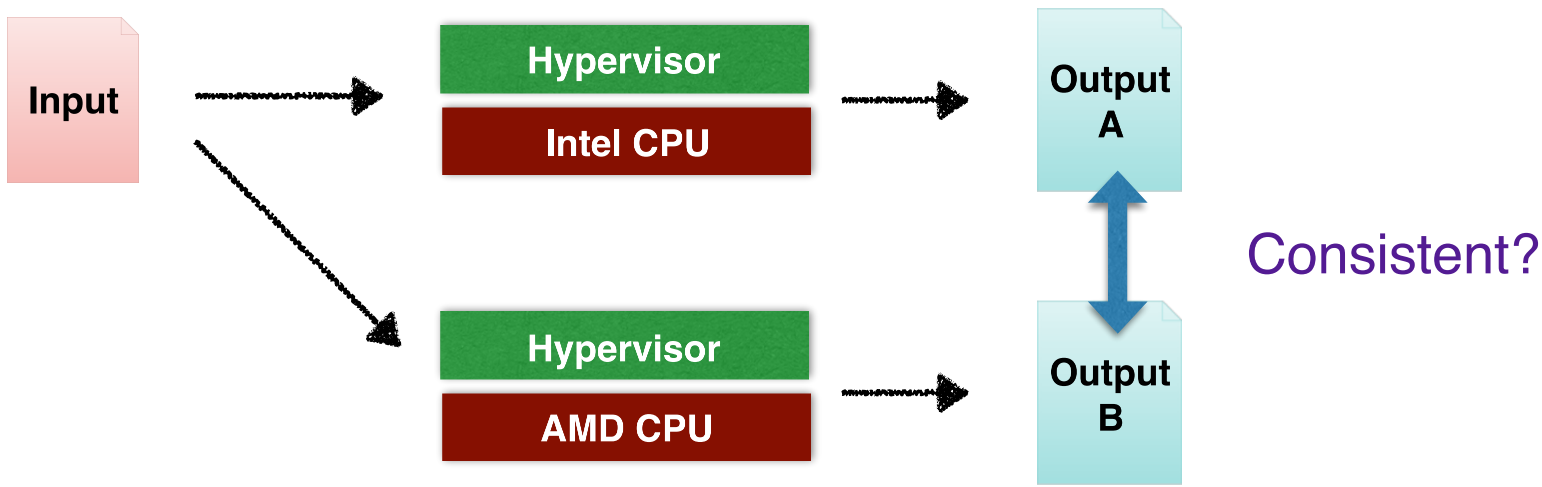
Challenge 2:
How to make symbolic execution scale?

- Traditional tests: Execute millions of VM instructions
- Key observation: Hypervisor interface allows externally setting the initial VM state

MultiNyx tests execute a **single VM instruction**

1. Set the initial VM state
2. Run a single VM instruction
3. Get the final VM state

Challenge 2:
How to make symbolic execution scale?

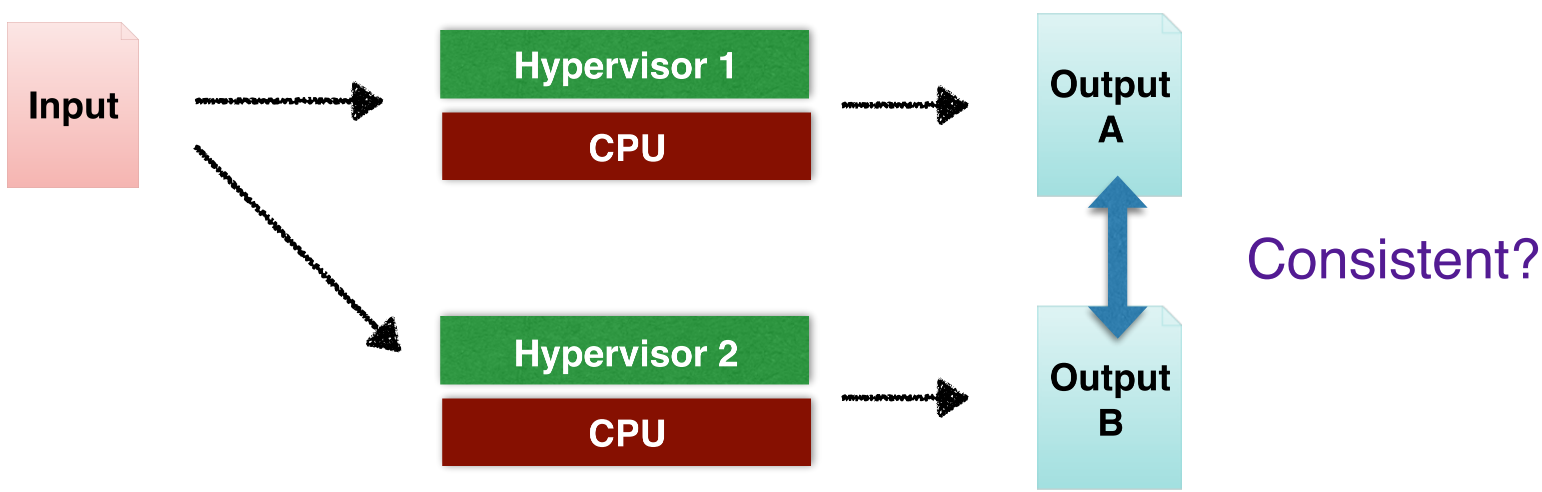


Run the same test on different configurations

What is the approach to solve the problem?

- 1. Systematically generate test cases using **symbolic execution (white-box testing)**
  - Challenge 1: How to model complex instructions?
  - Challenge 2: How to make symbolic execution scale?
- 2. Analyze test cases through **differential testing**

How to analyze the test results?



Use differential testing

How was MultiNyx implemented?

- Symbolic execution engine: Triton + Z3
- Executable specification: Bochs simulator

Component	Language	LOCs
KVM driver	C	2,400
KVM annotations	C	1,400
Low-level trace recording	C++	600
High-level trace recording	C++	1,300
Multi-level analysis	C++ / Python	3,100
Diff. testing and diagnosis	Bash / Python	4,400

How well does it work?

- +200,000 tests automatically generated for KVM
  - Took days to generate/analyze these tests
  - Symbolic execution is generally the most expensive part
- MultiNyx coverage is +8% higher than fuzzing
- MultiNyx tests revealed 739 mismatching tests

Example of KVM bug that was fixed

- Incorrect update of `%SP` register (2 bytes instead of 4 bytes)
  - And incorrect update of the VM memory
- Instruction **`PUSH %ES`**
  - EPT option disabled
  - Segment registers initialized with specific values
  - Execution in real mode

Let me know if you're interested in systems research

- How to test systems effectively?
- How to build effective sandboxes?
- What are the limitation of TEEs and how can we address them?
- How to ensure verification techniques work correctly?
- How to make containers lightweight?
- How to build good distributed system APIs?
- How to leverage and improve serverless computing?

Summary

- DS:
  - Concurrency, fault-tolerance
  - Safety an
- Systems research
- Next Tuesday: final class
  - Exam review
  - Optional class
  - What would you like to review?