CS 423
Operating System Design: Process VMs

Professor Michael Bailey
Spring 2018
Goals for Today

• **Learning Objective:**
  • Conclude discussion of virtualization w/ process VMs

• Announcements, etc:
  • MP3 due April 5th
  • Midterm scores released!
  • In-class discussion of tricky questions on Wednesday
  • More questions? Bates office hours are on Tue @ 11am

<——— Your instructor for the day

**Reminder:** Please put away devices at the start of class
Recap: Emulation/Interpretation

• Problem: Emulate guest ISA on host ISA
• Solution: Basic Interpretation

new

inst = code (PC)

opcode = extract_opcode (inst)

routineCase = dispatch (opcode)

jump routineCase

...

routineCase

call routine_address

jump new
Recap: Emulation vs. Binary Translation

• Emulation:
  – Guest code is traversed and instruction classes are mapped to routines that emulate them on the target architecture.

• Binary translation:
  – The entire program is translated into a binary of another architecture.
  – Each binary source instruction is emulated by some binary target instructions.
• Code discovery and dynamic translation
  – How to tell whether something is code or data?
  – Consider a jump instruction: Is the part that follows it code or data?

• Code location problem
  – How to map source program counter to target program counter?
  – Can we do this without having a table as long as the program for instruction-by-instruction mapping?
Recap: Dynamic Binary Translation

1. Start with SPC
2. Look up SPC → TPC in map table
3. Hit in Table?
   - Yes: Branch to TPC and execute block
   - No:
     - Translate new block
     - Store new SPC → TPC entry in table
4. Get SPC of next block
Process VMs

- Present the abstraction of a different machine and OS to a *process*. 

![Diagram showing Guest Process within Runtime, which is within Host OS]
Emulation Architecture

- Guest Memory Image
  - Loader
  - Initialization
  - Emulation Engine
    - Code Cache Manager
    - Code Cache
    - OS Call Emulation
    - Exception Emulation
  - Host OS
Note: Compatibility

- Process state equivalence at the point of interaction with the “external world”
  - When control transfers from guest process to host OS, state equivalence must hold
  - When control transfers back to guest process, state equivalence must hold (both of user managed and OS managed state)

- Consequences:
  - State does not need to be mapped correctly in between interactions with OS
Isomorphism Revisited

S1 \rightarrow S2 \rightarrow S3 \rightarrow S4

V(S1) \rightarrow V(S2) \rightarrow V(S3) \rightarrow V(S4)

S1' \rightarrow S2' \rightarrow S3' \rightarrow S4'

e(S1) \rightarrow e(S2) \rightarrow e(S3)

e'(S1) \rightarrow e'(S2) \rightarrow e'(S3)

Guest \rightarrow Host
State Mapping

• Guest registers ➔ Host registers/Memory
  – Guest context (and context switch)
  – Depends on who has more registers

• Memory address space mapping
  – Guest application (virtual) address space
    ➔ Host application (virtual) address space
Software translates guest to host virtual addresses. Disadvantage?
Direct Access Translation

Offset Translation

Guest Application Address Space

Mapped Guest Application Address Space

Runtime

Host Virtual Address Space

Direct Translation

Guest Application Address Space

Mapped Guest Application Address Space

Runtime

Host Virtual Address Space
Direct Access Translation

Offset Translation

Guest Application Address Space

Mapped Guest Application Address Space

Runtime

Host Virtual Address Space

Direct Translation

Guest Application Address Space

Host Virtual Address Space

Runtime

Mapped Guest Application Address Space

Limitations?
• Host OS Offers:
  – A system call to set memory protection (specifies page and access privileges)
  – A signal for a memory protection violation that can be delivered to the application (runtime)

• Memory protection
  – Each page has protection bits such as read/write or read/write/execute (e.g., you cannot execute data, or overwrite code)
  – What if guest architecture has read/write/execute protection whereas host has read/write only?
Page Size Issues

• What if page size on guest is a multiple of page size on host?
• What if page size on host is a multiple of page size on guest?
Page Size Issues

• What if page size on guest is a multiple of page size on host?
  – No problem. Just replicate page protection

• What if page size on host is a multiple of page size on guest?
  – Different guest pages mapped to same host page?
    • Problems?
  – Pad guest pages to size of host page?
    • Problems?
Page Size Issues

• What if page size on host is a multiple of page size on guest?
  – Different guest pages mapped to same host page? Problems?
    • What if pages have different protection?
    • Use the more conservative bits and handle violations accordingly
  – Pad guest pages to size of host page?
    • Makes address translation more difficult
Instruction Emulation

• Interpretation versus binary translation?
  – Interpretation:
    • no startup overhead
    • High overhead per instruction
  – Binary translation:
    • High startup overhead
    • Low overhead per instruction
  – Can we combine the best of both worlds?
• Interpretation versus binary translation?
  – Interpretation:
    • no startup overhead
    • High overhead per instruction
  – Binary translation:
    • High startup overhead
    • Low overhead per instruction
  – Can we combine the best of both worlds?
    • Small program: Do interpretation
    • Large program: Do binary translation
• Initially assume small program
  – Do Interpretation
• Count the number of times each block is executed
• If a block is executed more than N times, do binary translation on this block
Interrupts Emulation

• Two types:
  – Traps (caused by instructions in the program)
  – Hardware interrupts (caused by asynchronous external events)

• For Traps and Exceptions:
  – Ensure that all instructions prior to trap have been executed
  – Ensure that none of the instructions after the trap have been executed

• For Interrupts:
  – Emulated code must be in interruptible state
• How to detect them?
  – Both guest and host support same trap (e.g., page fault). Map guest trap to host trap: capture trap signal, execute the translated guest handler
  • Runtime intercepts all signals and handles them
  – Guest supports trap/exception that host does not support (or does not deliver to the application). Check for exception conditions in the emulated software explicitly
• When an interrupt occurs:
  – Interpretation: When an interrupt occurs, finish interpreting the current instruction and execute the interrupt handler
  – Binary translation: When an interrupt occurs, the emulated code may be in non-interruptible state (what does that mean?)
    • Need well-defined boundaries where emulated code is interruptible.
    • What is a suitable boundary?
    • When interrupt occurs, execute emulated guest code until boundary is reached, then execute the interrupt handler.
• When an interrupt occurs:
  – Interpretation: When an interrupt occurs, finish interpreting the current instruction and execute the interrupt handler
  – Binary translation: When an interrupt occurs, the emulated code may be in non-interruptible state (what does that mean?)
    • Need well-defined boundaries where emulated code is interruptible.
    • What is a suitable boundary? **BLOCK BOUNDARIES**
    • When interrupt occurs, execute emulated guest code until boundary is reached, then execute the interrupt handler.
• When an interrupt occurs:
  – Interpretation: When an interrupt occurs, finish interpreting the current instruction and execute the interrupt handler
  – Binary translation: When an interrupt occurs, the emulated code may be in non-interruptible state (what does that mean?)
    • Need well-defined boundaries where emulated code is interruptible.
    • What is a suitable boundary? **BLOCK BOUNDARIES**
    • When interrupt occurs, execute emulated guest code until boundary is reached, then execute the interrupt handler.

What if blocks are chained?
• When an interrupt occurs, the emulated code may be in non-interruptible state
  – Determine which block is currently running
  – Unchain the block from the next by replacing the jump at the end of the block to a transfer of control to the emulation manager.
  – Let the block finish
  – Control is transferred to emulation manager which invoked interrupt handler.
Note: Guest OS Emu

- Does not have to translate guest OS instructions one a time
  - Translate entire functions into equivalent ones
  - Example: replace a disk I/O system call on the guest with an equivalent disk I/O call on the host

- Not all guest system calls need to be translated to host calls; some are handled by the runtime.
  - Example: Calls installing a new signal handler may be handled by the runtime since runtime intercepts all signals and maintains their handlers.

- Generally an ad hoc process (case-by-case).