LECTURE

VMM

⇒ virtual machine monitors

Idea: Apps / OS ⇒ Apps Apps / OS OS / VMM ← very thin layer, simulate machine among OS's

why useful?
- ⇒ many OS's
- ⇒ consolidation (cost)

⇒ Ancient History: IBM mainframes in 70's
⇒ Modern History: Disco @ Stanford (Rosenblum)
- ⇒ virtual CPU
- ⇒ virtual Memory

⇒ virtual CPU

App / OS / VMM

how to virtualize CPU? (how to limit + control execution? ⇒ apps, OS ✗)

classic:

@ Boot: (kernel mode)

OS runs first,
installs interrupt + trap handlers (priv. inst.)
- ⇒ sys calls
- ⇒ timer + other interrupts

} how we set up limited direct execution

:

@ Process runs (user mode)
issues system call (trap inst.)

trap ↘ ↗ rett

OS trap handler

with VMM:

App / OS / VMM

@ Boot
VMM runs first
trap handlers installed
then OS "boots": (what mode to run it in? (user, kernel?))
tries to install trap handlers
what happens? (hint: illegal inst. trap)
VMM records where OS trap handlers are

can't be full priv. mode

:

@ Process runs
..... user mode
issues sys call: trap (what happens?)

trap ↘ ↗ OS trap handler ... ↘ rett (illegal) / rett ↗

VMM (Disco) VMM

Note: Disco allows OS to "handle" sys call w/o knowing anything about sys call!

⇒ Virtual Memory

V → P → Machine

VMM: some kind of "page table" per OS
(OS: still has page table/process)

IRIX / MIPS: s/w managed TLB

classic:
OS @ Boot: install TLB miss handler
proc @ runtime:
mem ref:
- → TLB miss ↘ OS → lookup PFN in PT, install translation into TLB
- ↗ rett
- retry, TLB hit →

with VMM:
@ boot VMM: installs TLB miss handler
OS: "installs" TLB miss handler address (trap, Disco/VMM records it)

@ run:
Proc: (user mode)

where does OS data live?

TLB miss → OS TLB miss handler → lookup PT → install in TLB → rett → retry

illegal → VMM: OS wants V→P, VMM wants P→M, install V→M

VMM:?

time overhead: even worse!

Other issues: lack of info
- ⇒ idle loop
- ⇒ zero'd pages
- ⇒ memory filled w/ redundant copies of info (think OS code)

And now: one of the problems of virtualization: overhead (time)
(OS traps take longer)