Lecture #6 CS695 25. 1. 2023

⊛ interrupt handling (with VMs & VMMs)

- interrupts / traps
  - external / IO interrupt — phy. signaling to the CPU
  - exception
  - ⊛ explicit interrupt } s/w → pg. flt, null deref.
    - int

⊛ Native systems int processing

int 0x80 → user space

system call

h/w:
- index in IDT using the interrupt vector #
- check PLs for integrity
- save ESP, SS → to internal temp regs of the CPU

LOAD ESP } this is
LOAD SS } Kernel stack ‖ TSS

SAVE USER ESP
USER SS
USER eflags
user CS
USER EIP ——— jump to ISR

- save context & trap
- s/w privileges
- ~~load~~ ISR / jump to ISR
- checks for call integrity

took
open
↑
wrappers for system calls.
eax ←
ebx ←
int 0x80h

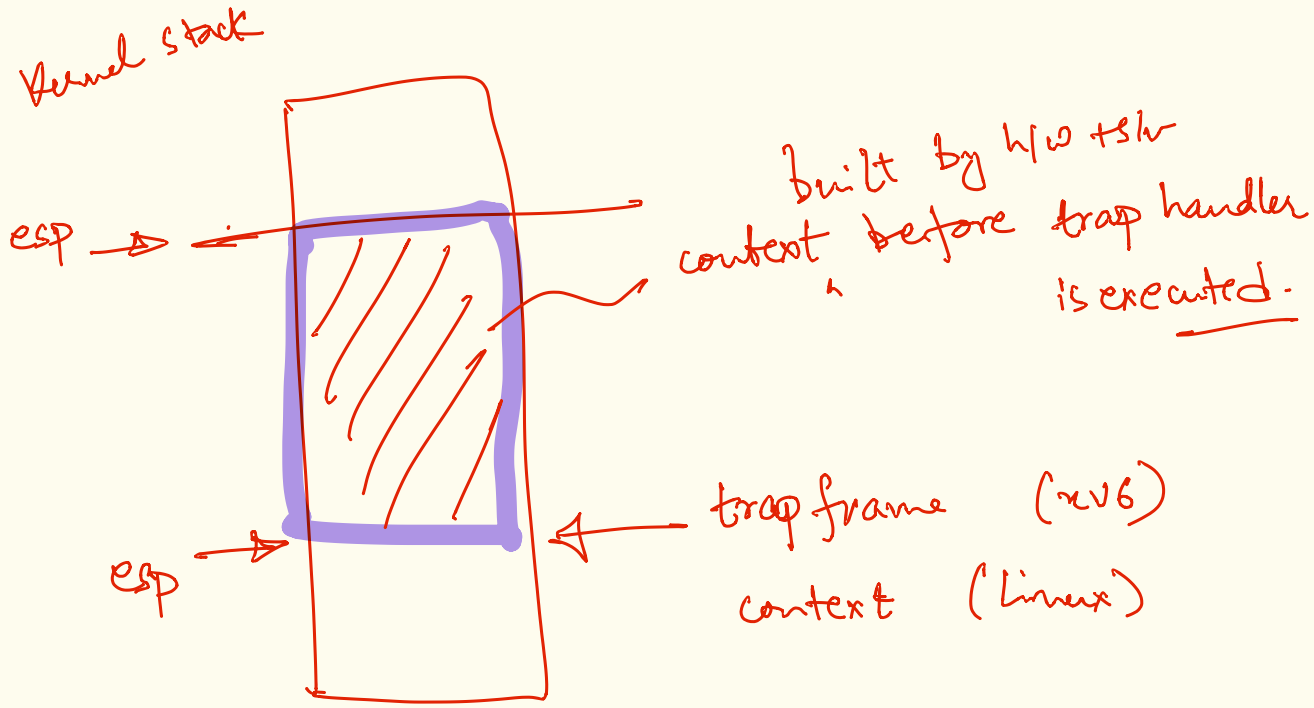
s/w:
- push error code
- push trap/intr. no } trap info.
- push ds
  gs
  es
  fs } other segments
- push eax
  bx
  cx
  dx } general purpose regs.

⊛ arguments info

jmp idt_table [trap no.] — array of fp's. — system call handler

```
system-call-handler( ) {
    fn = sys-call-table[eax];
    call fn;
    iret;
}
```

sys call no.



return path in the ISR

- pop al
  pop ds
  pop gs
  pop es
  pop fs
  adjust ESP ~ to move over error code & trap no.
  iret
  - h/w-assisted instruction.
  - rolls back ESP of kernel
  - loads user regs.
  - jmps to CS:EIP w/ PL=3

(2) what happens to system calls from VMs ?

(i) full virtualized VMs. ~ binary translation / BTVM

- on interrupt
  - control in VMM
  - context on VMM stack
  - ISR uses context to determine that this int^n should be handled by guest.
  - emulate an interrupt
    - copying context from vmm to guest Kernel stack ? ~ update to ESP & SS
    - jmp to guest at CS:EIP of guest ISR.

~ control to VMM

~ VMM needs to IRET interrupt to guest

~ guest executes handler

~ iret in guest traps to VMM

- VMM returns to guest user space

(ii) Similar in PV, but via hypercalls for critical state interrupts with h/w assisted VMs.

Ring 3 — Ring 3 — appln. } system call

VMM — Ring 0 — Ring 0 — guestOS

root — non-root

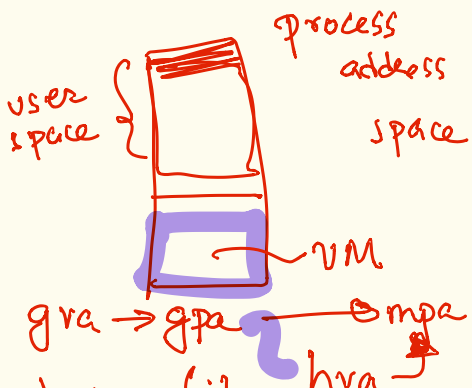
← vmexit

~ depends on exit conditions & IDT setup.

Assignment #3

~ build a simple VM with KVM ~ LKM → Type 2

use



(i) qemu + kvm
↳ user process, emulator

(ii) our soln. will look like qemu

(iii) /dev/kvm ~ device file

```
fd = open("/dev/kvm");
ioctl(fd, params)
```

ioctl ~ generic system call.

action, arguments