CS 61 Lecture 13 — October 15, 2019

Protected Control Transfers (exception mechanisms)

- mechanism to give kernel control at a safe code location in a safe environment

Types

1. traps: 'syscall' instruction, 'int 3'
   - voluntarily caused by unprivileged software
   - calling convention (can have a)
2. interrupts: timer interrupt
   - involuntarily caused by hardware
   - no calling convention, so must appear transparent, all registers must be restored
3. faults: segmentation (page) fault
   - involuntary, caused by software error *
   - no calling convention, save + restore registers

* why they're caused, whether caused by unprivileged software or involuntarily, and how hardware mechanisms to control transfer is restricted

PCT requirements

- cannot cause errors on its own
- configure code location / environment
- registers / hardware features define entry points into kernel
  - must be privileged
- kernel memory protection

- changing memory layout (memory configuration) is privileged.

Protected Control Transfer Examples

sys_getpid (v-lib.hh file) — trap

- calls make_syscall(PID)
  - has way to move assembly to a C++ program
  - sets up registers / calling convention
    - i/o registers
    - input registers
    - modifiable registers
      - caller-saved registers for syscalls
      - only convert general purpose, not special purpose registers
  - rax holds syscall # on input
    after syscall, rax holds return value

PCT what changes?

- always changes
  - hardware saves process' values
    - %rip : runs kernel instructions
    - %cs : kernel privilege register
      - for interrupts, saved in memory
      - general purpose
        for syscalls, swapped into other registers
  - %rflags, %rsp ← x86-64 specific

* no way to "fake" a protected control transfer

* sudo has privilege 3(even programs running as root don't have machine privilege, only programs in kernel do

* modern processors have in-built support for VMs w/ multiple layers of privilege. You can always emulate privilege.

* faults can happen in kernel mode or unprivileged mode

Page Table

- provides virtual memory protection and flexible address mappings
- instructions refer to VIRTUAL addr —(kernel enforced mapping)→ physical addr
- x86-64:
  - %cr3 register = addr of current page table. (must be a physical address)
    is a privileged register. (stored in memory)

One-Level Page Table

index	
0	0x3000 | PTE_P
1	0xFF000 | PTE_P | PTE_W
2	0x7000 | PTE_P | PTE_W | PTE_U
3	0

what physical addr does va 0x21F6 map to? pa 0x71F6

(2nd index)

constants, assume PTE_P | (PTE_W, PTE_U)*
(4) (2)

```
mem(va, flags):
    assert(flags & PTE_P);
    pt = %cr3;   get page table
```

```
off = (va & 0xFFF);   calc offset
indx = va >> 12;      calc index
e = pt[indx];         look up entry @ index
if((e&flags)!=flags)  check if entry allows
                      indx
  {fault;}
return (e & ~0xFFF) | off;   mask off flags, add offset
```

every memory access requires present flag

* Bitwise arithmetic
  - set bits = 0
    e = 0x3001
    & ~0xFFF ≡ & 0xFFFF'FFFF'FFFF'F000

x86-64 4-level page table