Operating Systems Engineering

Recitation 3: Interrupts, Exceptions

Based on MIT 6.828 (2012, lec5)
Stacks in the System

- Where can our code run?
  - User mode
  - Kernel Mode

- Same stack for user and kernel?
  - How many user stacks? 1 or many?
  - 1 for each process? More?
  - How many kernel stacks?

- Why would you need more than 1?
Mode Transitions

• What mode transitions do we have?
• User – Kernel
  − System calls
  − Interrupts
  − Exceptions
• Kernel - User
• Kernel - Kernel
  − Interrupts
  − Context switch
Mode Transitions

• Why is user - kernel transition delicate?

• What do we need to maintain?
  – Need to maintain isolation
  – Need to maintain transparency
Reminder: x86 Privilege Levels

• Called CPL in x86
  • Bottom two bits of the cs register

CPL:

• CPL=0 – kernel mode – privileged
• CPL=3 – user mode – not privileged
System Calls

- What needs to happen in a system call?
  1. Save user state
  2. Transition to kernel (stack, CPL=0)
  3. Choose kernel entry point
  4. Get system call arguments

- Is it secure? What if user could interfere between 2 and 3?
Interrupt vectors

• Where does “int $0x30” jump to?
  – $0x30 is an interrupt vector
  – A vector is an allowed kernel entry point
• x86 has 256 vectors
• different uses (devices, exceptions, syscalls…)
• Each vector in an index in the IDT (Interrupt Descriptor Table)
Interrupt Descriptor Table

- Table of all interrupt descriptors
- Pointer to by the **IDTR** register
- Contains interrupt / trap gates

```
  31  16  15  14  13  12  8  7  5  4  0
  |   |   |   |   |   |   |   |   |   |   |
  | Offset 31..16 | P | D | L | O | D | I | L | T | Y | P |
  31  16  15  0
  | Segment selector | Offset 15..0 |

Type: 0=Interrupt gate, 1=Trap gate
Selector: Destination CS
Offset: Destination IP or EIP

P Present
DPL Descriptor privilege level (CPL required to invoke gate)
D Size of gate (0=16-bits, 1=32-bits)```
Interrupt Handling Flow

- Fetch vector's descriptor from IDT
- If segment selector's PL < CPL
  - load SS and ESP from TSS
- Push user SS
- Push user ESP
- Push user EFLAGS
- Push user CS
- Push user EIP
- Clear some EFLAGS bits
- Set CS and EIP from IDT
Interrupt Handling in xv6

- Entry points – vectors.pl (sheet 29)
- Kernel SS and ESP setup
  - switchuvvm() in vm.c (sheet 17)
- IDT setup
  - tvinit() in trap.c (sheet 30)
- Entry point calls alltraps(), which calls trap()
- Who initialized and passed *trapframe* to trap()? 
  - *trapframe* (sheet 06)
struct trapframe {
    uint edi;
    uint esi;
    uint ebp;
    uint oesp;
    uint ebx;
    uint edx;
    uint ecx;
    uint eax;

    ushort gs; ushort padding1;
    ushort fs; ushort padding2;
    ushort es; ushort padding3;
    ushort ds; ushort padding4;
    uint trapno;

    uint err;
    uint eip;
    ushort cs;
    ushort padding5;
    uint eflags;

    uint esp;
    ushort ss;
    ushort padding6;
};
System Call Handling

• If tf->trapno == T_SYSCALL (0x40) – trap() calls syscall() (sheet 33)
• Syscall number determined by tf->eax
• Where is this value set?
• No value returned to trap – where is system call return value?
• How are parameters passed?
Handling Other Interrupts

- Other interrupts are handled the same
- Can be handled internally by kernel
- User can subscribe to some POSIX Signals
- Can happen in kernel
- Trapframe is a bit different, how?
Device Interrupts

- Hardware generated
- Different vector for different devices
  - Timer
  - Console
  - Disk
  - Network
Fork

fork() – proc.c (sheet 23)
allocproc() – proc.c (sheet 22)