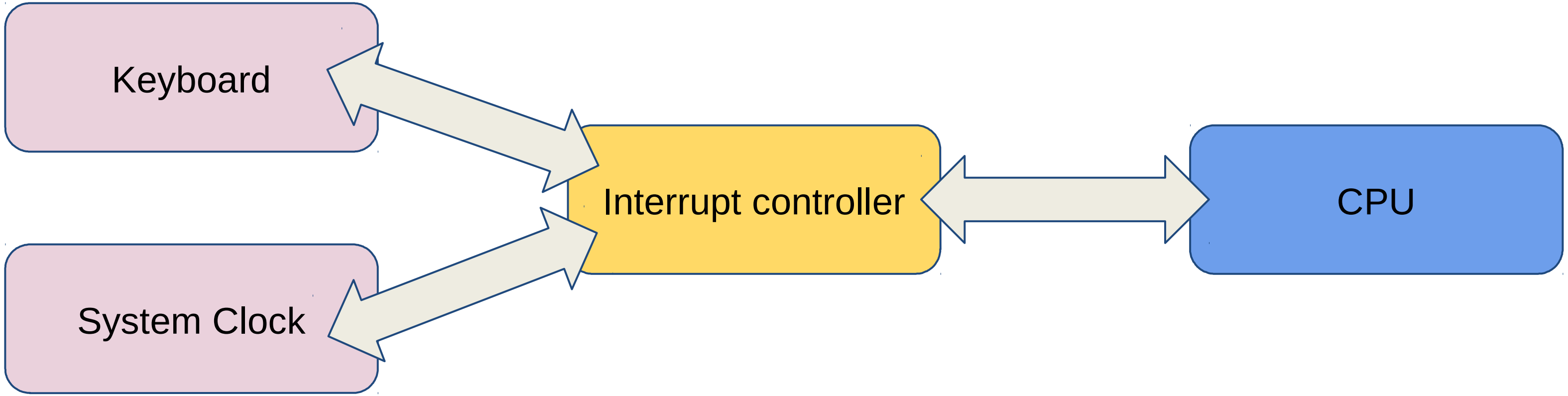
Interrupt Management

We are interrupting this presentation so we can have a quick chat about x86 interrupt management.

High level view of the system



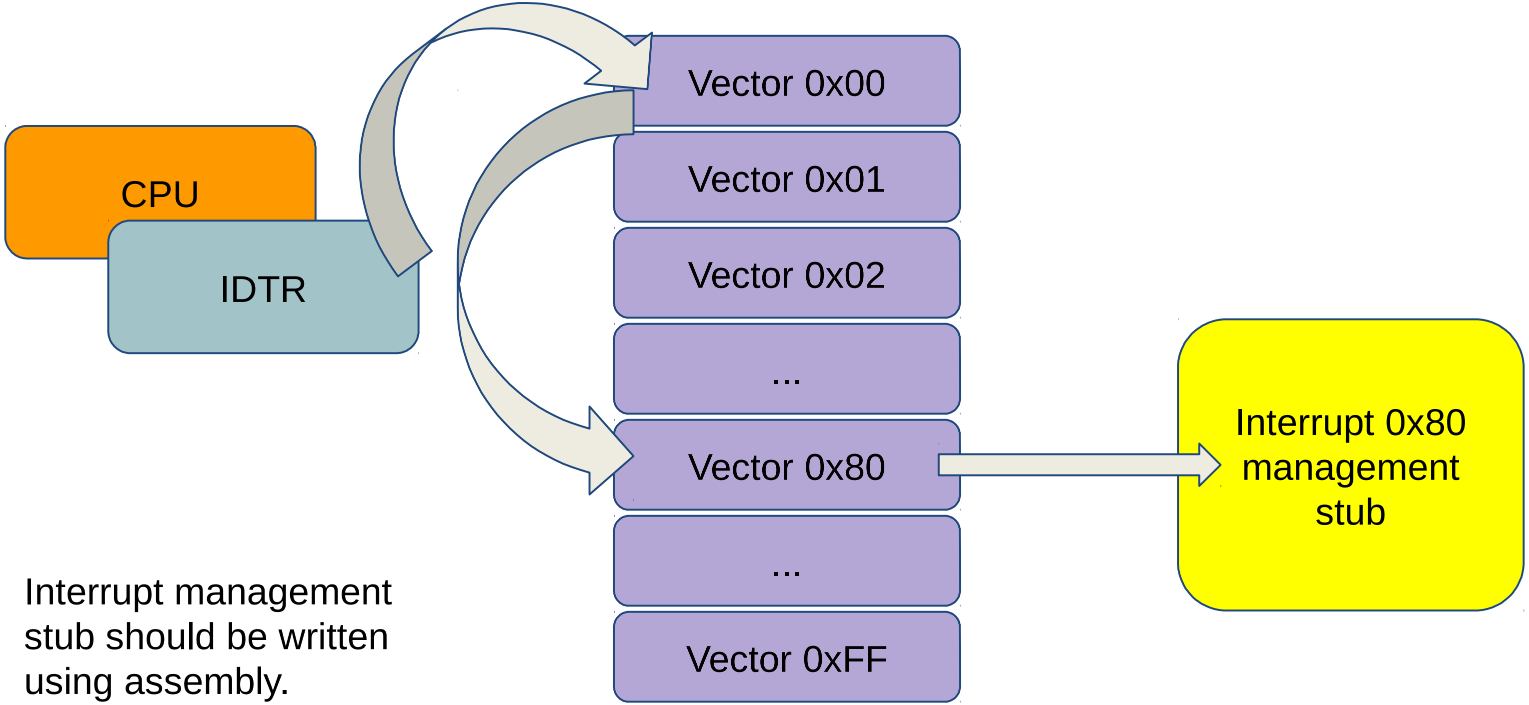
Keyboard

System Clock

Interrupt controller

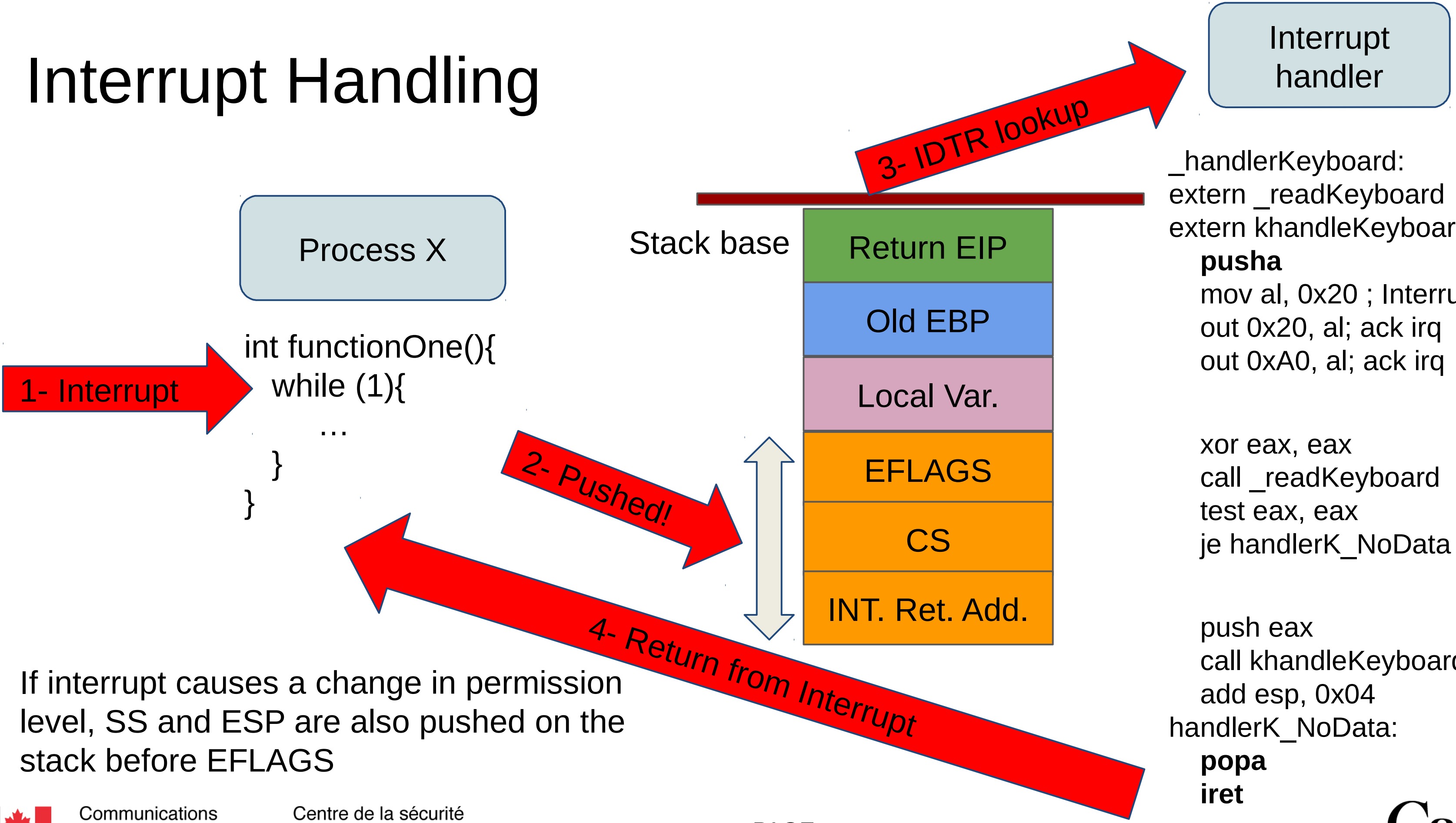
CPU

Interrupt delivery to the CPU



Interrupt management stub should be written using assembly.

Interrupt Handling



```
int functionOne(){
  while (1){
      …
  }
}
```

```
_handlerKeyboard:
extern _readKeyboard
extern khandleKeyboard
    pusha
    mov al, 0x20 ; Interrupt ACK
    out 0x20, al; ack irq
    out 0xA0, al; ack irq

    xor eax, eax
    call _readKeyboard
    test eax, eax
    je handlerK_NoData

    push eax
    call khandleKeyboard
    add esp, 0x04
handlerK_NoData:
    popa
    iret
```

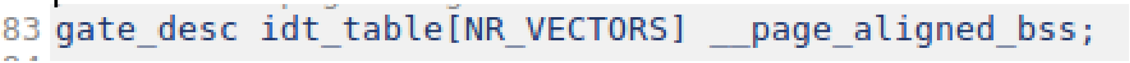
If interrupt causes a change in permission level, SS and ESP are also pushed on the stack before EFLAGS

x86 Exceptions

Exception	Description
0	Divide error
6	Invalid opcode
12	Stack segment fault
14	Page fault
17	Alignement check
19	SIMD floating point exception
...	...

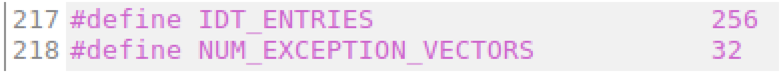
On Linux

traps.c



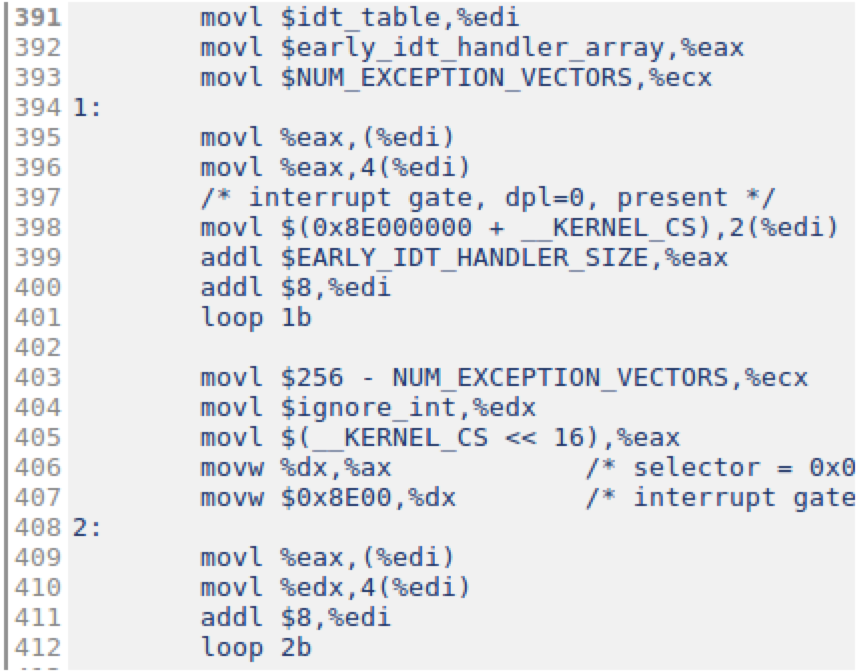
```
gate_desc idt_table[NR_VECTORS] __page_aligned_bss;
```

segment.h



```
#define IDT_ENTRIES                     256
#define NUM_EXCEPTION_VECTORS           32
```

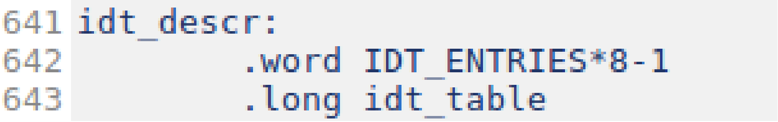
head_32.s



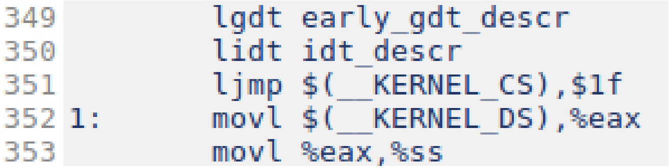
```
        movl $idt_table,%edi
        movl $early_idt_handler_array,%eax
        movl $NUM_EXCEPTION_VECTORS,%ecx
1:
        movl %eax,(%edi)
        movl %eax,4(%edi)
        /* interrupt gate, dpl=0, present */
        movl $(0x8E000000 + __KERNEL_CS),2(%edi)
        addl $EARLY_IDT_HANDLER_SIZE,%eax
        addl $8,%edi
        loop 1b

        movl $256 - NUM_EXCEPTION_VECTORS,%ecx
        movl $ignore_int,%edx
        movl $(__KERNEL_CS << 16),%eax
        movw %dx,%ax            /* selector = 0x0
        movw $0x8E00,%dx        /* interrupt gate
2:
        movl %eax,(%edi)
        movl %edx,4(%edi)
        addl $8,%edi
        loop 2b
```

head_32.s



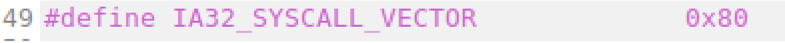
```
idt_descr:
        .word IDT_ENTRIES*8-1
        .long idt_table
```



```
        lgdt early_gdt_descr
        lidt idt_descr
        ljmp $(__KERNEL_CS),$1f
1:      movl $(__KERNEL_DS),%eax
        movl %eax,%ss
```

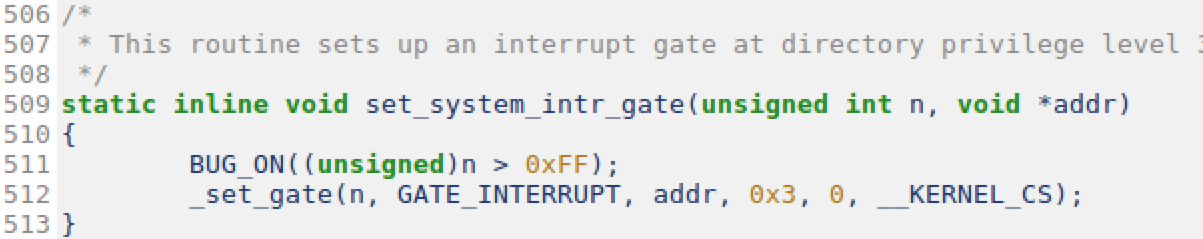
On Linux

irq_vectors.h



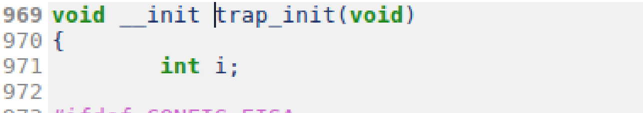
```
#define IA32_SYSCALL_VECTOR             0x80
```

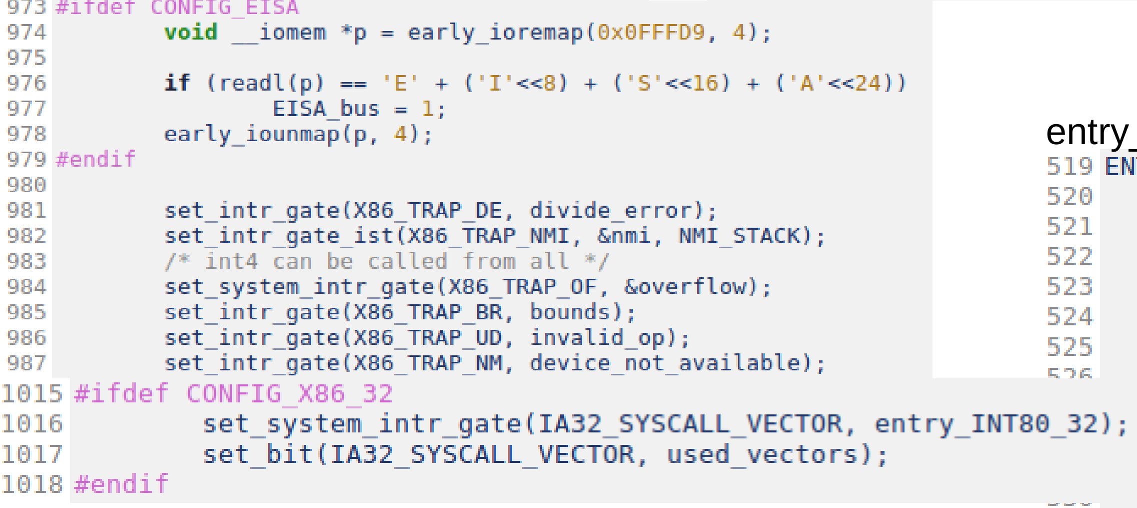
desc.h



```
/*
 * This routine sets up an interrupt gate at directory privilege level
 */
static inline void set_system_intr_gate(unsigned int n, void *addr)
{
        BUG_ON((unsigned)n > 0xFF);
        _set_gate(n, GATE_INTERRUPT, addr, 0x3, 0, __KERNEL_CS);
}
```

traps.c





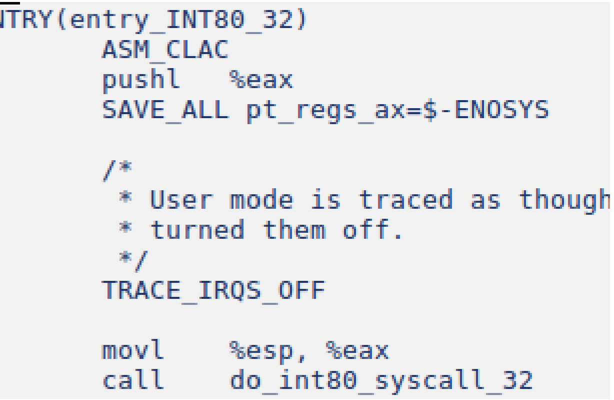
```
void __init trap_init(void)
{
        int i;

#ifdef CONFIG_EISA
        void __iomem *p = early_ioremap(0x0FFFD9, 4);

        if (readl(p) == 'E' + ('I'<<8) + ('S'<<16) + ('A'<<24))
                EISA_bus = 1;
        early_iounmap(p, 4);
#endif

        set_intr_gate(X86_TRAP_DE, divide_error);
        set_intr_gate_ist(X86_TRAP_NMI, &nmi, NMI_STACK);
        /* int4 can be called from all */
        set_system_intr_gate(X86_TRAP_OF, &overflow);
        set_intr_gate(X86_TRAP_BR, bounds);
        set_intr_gate(X86_TRAP_UD, invalid_op);
        set_intr_gate(X86_TRAP_NM, device_not_available);
#ifdef CONFIG_X86_32
        set_system_intr_gate(IA32_SYSCALL_VECTOR, entry_INT80_32);
        set_bit(IA32_SYSCALL_VECTOR, used_vectors);
#endif
```

entry_32.s



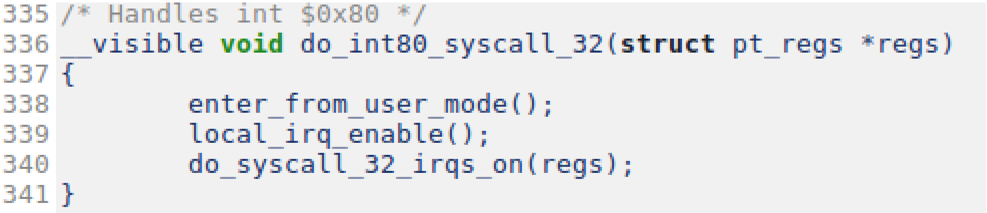
```
ENTRY(entry_INT80_32)
        ASM_CLAC
        pushl   %eax
        SAVE_ALL pt_regs_ax=$-ENOSYS

        /*
         * User mode is traced as though
         * turned them off.
         */
        TRACE_IRQS_OFF

        movl    %esp, %eax
        call    do_int80_syscall_32
```

On Linux

common.c



```c
/* Handles int $0x80 */
__visible void do_int80_syscall_32(struct pt_regs *regs)
{
        enter_from_user_mode();
        local_irq_enable();
        do_syscall_32_irqs_on(regs);
}
```

You are free to continue your exploration from here :)
The point is simply to make you understand a little bit of what happens when a syscall is made on Linux before moving one to understanding how to do a syscall using assembly programming.