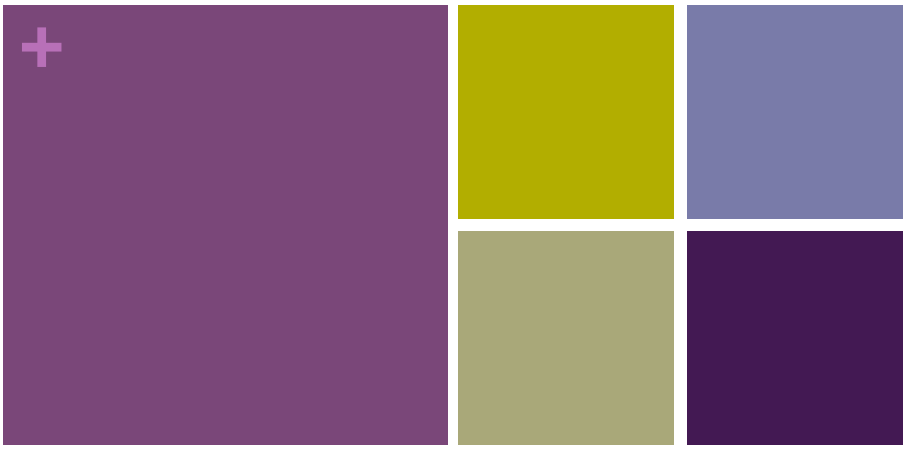


Projects: Developing an OS Kernel for x86

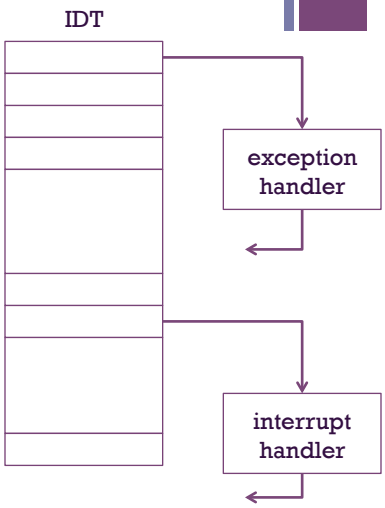
Low-Level x86 Programming: Exceptions, Interrupts, and Timers

Overview

- Handling Intel Processor Exceptions: the Interrupt Descriptor Table (IDT)
- Handling Hardware Device Interrupts the old fashioned style: IRQs and the 8259 Programmable Interrupt Controller.
- Example Interrupt Handlers:
  - The Programmable Interval Timer
- References: www.osdever.net
  - Brandon Friesen's beginner OS development tutorial by warmaster199.
  - others …

Interrupt Descriptor Table

- Interrupt Vector Table x86-style:
  - processor exceptions, hardware interrupts, software interrupts
- 256 entries: Each entry contains address (segment id and offset) of interrupt handler (interrupt service routine).
- The first 32 entries reserved for processor exceptions (division by zero, page fault, etc.)
- Hardware interrupts can be mapped to any of the other entries using the Programmable Interrupt Controller (e.g. 8259 PIC, see later)



Processor Exceptions

Exception #	Description	Error Code?
0	Division By Zero Exception	No
1	Debug Exception	No
2	Non Maskable Interrupt Exception	No
3	Breakpoint Exception	No
4	Into Detected Overflow Exception	No
5	Out of Bounds Exception	No
6	Invalid Opcode Exception	No
7	No Coprocessor Exception	No
8	Double Fault Exception	Yes
9	Coproc. Segment Overrun Exception	No
10	TSS Exception	Yes
11	Segment Not Present Exception	Yes
12	Stack Fault Exception	Yes
13	General Protection Fault Exception	Yes
14	Page Fault Exception	Yes
15	Unknown Interrupt Exception	No
16	Coprocessor Fault Exception	No
17	Alignment Check Exception (486+)	No
18	Machine Check Exception (Pentium/586+)	No
19 to 31	Reserved Exceptions	No

Handling Interrupts/Exceptions

```
global _isr0
global _isr1
global _isr2

...

global _isr8

…

global _isr30
global _isr31
```

```
;  0: Divide By Zero Exception
_isr0:
   cli
   push byte 0    ; push dummy
   push byte 0    ; push interrupt no
   jmp isr_common_stub
```

```
;  8: Double Fault Exception (Error Code!)
_isr8:
   cli
   ; don't push dummy!
   push byte 8
   jmp isr_common_stub
```

```
extern _dispatch_exception
; The common ISR stub.
isr_common_stub:
   pusha
   push ds
   push es
   push fs
   push gs
   mov ax, 0x10
   mov ds, ax
   mov es, ax
   mov fs, ax
   mov gs, ax
   mov eax, esp
   push eax
   mov eax, _dispatch_exc
   call eax
   pop eax
   pop gs
   pop fs
   pop es
   pop ds
   popa
   add esp, 8 ; pop code and int no
   iret
```

```
/* This defines what the stack looks like when the exception/interrupt
   reaches the exception dispatcher. */
typedef struct regs
{
   unsigned int gs, fs, es, ds;         /* pushed in common stub */
   unsigned int edi, esi, ebp, esp,
                ebx, edx, ecx, eax;     /* pushed by pusha */
   unsigned int int_no, err_code;               /* pushed in _isrXX */
   unsigned int eip, cs, eflags, useresp, ss;  /* pushed by processor */
} REGS;
```

High-Level Int/Exc Handler

```
extern _dispatch_exception
; The common ISR stub.
isr_common_stub:
   pusha
   push ds
   push es
   push fs
   push gs
   mov ax, 0x10
   mov ds, ax
   mov es, ax
   mov fs, ax
   mov gs, ax
   mov eax, esp
   push eax
   mov eax, _dispatch_exception
   call eax
   pop eax
   pop gs
   pop fs
   pop es
   pop ds
   popa
   add esp, 8 ; pop code and int no
   iret
```

```
/* Exception handlers are functions that take a pointer to
   a REGS structure as input and return void. */
typedef void (* ExceptionHandler)(REGS*);

/* List of registered exception handlers. */
static ExceptionHandler handler_table[EXCEPTION_TABLE_SIZE];
```

```
void dispatch_exception(REGS * _r) {

  unsigned int exc_no = _r->int_no; /* get exception no */

  ExceptionHandler handler = handler_table[exc_no];

  if (handler)      /* Is a handle registered? */
    handler(_r);    /* If so, fire it up! */
}
```

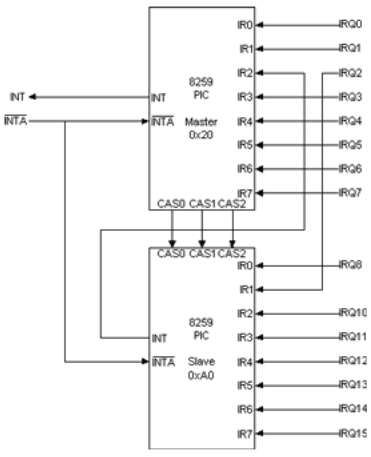
Initializing the IDT

```
extern void isr0();
extern void isr1();
/*…*/
extern void isr31();

void init_exception_dispatcher() {
  /*   Add any new ISRs to the IDT here. */
  idt_set_gate( 0, (unsigned) isr0, 0x08, 0x8E);
  idt_set_gate( 1, (unsigned) isr1, 0x08, 0x8E);
  /* … */
  idt_set_gate(31, (unsigned)isr31, 0x08, 0x8E);

  /*   Initialize high-level exception handler */
  for(int i = 0; i < EXCEPTION_TABLE_SIZE; i++) {
    handler_table[i] = NULL;
  }
}
```

For details of "idt_set_gate()" and installation of IDT check out the code (file "idt.C").

Hardware Interrupts: The Programmable Interrupt Controller (PIC)

- Interrupt control on PC/AT by cascaded (master/slave) pair of 8259 PICs.
- Interrupts (IRQs) are mapped to IDT entries. (Typically inconvenient ones, can be remapped.)
- Interrupt service routine must send EOI (end-of-interrupt) signal to PIC (to both PICs if interrupt from slave)



Assigned Interrupt Lines in the PIC

- Master 8259
  - IRQ0 – Intel 8253 or Intel 8254 PIT, aka the system timer
  - IRQ1 – Intel 8042 keyboard controller
  - IRQ2 – not assigned in PC/XT; cascaded to slave 8259 INT line in PC/AT
  - IRQ3 – 8250 UART serial port COM2 and COM4
  - IRQ4 – 8250 UART serial port COM1 and COM3
  - IRQ5 – hard disk controller in PC/XT; Intel 8255 parallel port LPT2 in PC/AT
  - IRQ6 – Intel 82072A floppy disk controller
  - IRQ7 – Intel 8255 parallel port LPT1 / spurious interrupt
- Slave 8259 (PC/AT and later only)
  - IRQ8 – real-time clock (RTC)
  - IRQ9 – no common assignment
  - IRQ10 – no common assignment
  - IRQ11 – no common assignment
  - IRQ12 – Intel 8042 PS/2 mouse controller
  - IRQ13 – math coprocessor
  - IRQ14 – hard disk controller 1
  - IRQ15 – hard disk controller 2

Handling Hardware Interrupts

```
global _irq0
/*… */
global _irq15

; 32: IRQ0 – start at IDT entry 32
_irq0:
   cli
   push byte 0
   push byte 32
   jmp irq_common_stub

; 33: IRQ1
_irq1:
   cli
   push byte 0
   push byte 33
   jmp irq_common_stub
```

```
extern _dispatch_interrupt
/* similar to ISRs */
irq_common_stub:
   pusha
   push ds
   push es
   push fs
   push gs
   mov ax, 0x10
   mov ds, ax
   mov es, ax
   mov fs, ax
   mov gs, ax
   mov eax, esp
   push eax
   mov eax, _dispatch_interrupt
   call eax
   pop eax
   pop gs
   pop fs
   pop es
   pop ds
   popa
   add esp, 8
   iret
```

Hardware Interrupts: High Level

```
extern _dispatch_interrupt
/* similar to ISRs */
irq_common_stub:
    pusha
    push ds
    push es
    push fs
    push gs
    mov ax, 0x10
    mov ds, ax
    mov es, ax
    mov fs, ax
    mov gs, ax
    mov eax, esp
    push eax
    mov eax, _dispatch_interr
    call eax
    pop eax
    pop gs
    pop fs
    pop es
    pop ds
    popa
    add esp, 8
    iret
```

```
void dispatch_interrupt(REGS * _r) {

    unsigned int int_no = _r->int_no - IRQ_BASE;
    InterruptHandler handler = handler_table[int_no];

    if (!handler) {
        /* --- NO DEFAULT HANDLER REGISTERED. */
        abort();
     }

    handler(_r);

    /* This is an interrupt that was raised by the interrupt controller. We need
        to send and end-of-interrupt (EOI) signal to the controller after the
        interrupt has been handled. */
    /* Check if the interrupt was generated by the slave interrupt controller.
        If so, send an End-of-Interrupt (EOI) message to the slave controller. */
    if (generated_by_slave_PIC(int_no)) { /* i.e. int_no < 8 */
        outportb(0xA0, 0x20);
    }
    /* Send an EOI message to the master interrupt controller. */
    outportb(0x20, 0x20);
}
```

Example: Periodic Timer

```
/* timer interrupt handler*/
void SimpleTimer::handler(REGS *r) {
    /* Increment our "ticks" count */
    ticks++;
    /* Whenever a second is over, we update counter. */
    if (ticks >= hz ) {
        seconds++;
        ticks = 0;
        Console::puts("One second has passed\n");
    }
}

/* Set the interrupt frequency for the simple timer.
/* Preferably set this before installing the timer handler!
void SimpleTimer::set_frequency(int _hz) {
  hz = _hz;                          /* Remember the frequen
  int divisor = 1193180 / _hz;  /* The input clock runs at
  outportb(0x43, 0x34);          /* Set command byte to b
  outportb(0x40, divisor & 0xFF); /* Set low byte of divis
  outportb(0x40, divisor >> 8);      /* Set high byte of divi
}
```

```
void main() {

    GDT::init();
    Console::init();
    IDT::init();
    init_exception_dispatcher();
    IRQ::init();
    init_interrupt_dispatcher();

    SimpleTimer::init();
    SimpleTimer::set_frequency(100);
    /* timer ticks to 10ms. */
    reg_int_handler(0, SimpleTimer::handler);

    __asm__ __volatile__ ("sti");

    Console::puts("Hello World!\n");

    for (;;);
}
```