ARRANQUE DE MINIX

mpx386.s

Raúl F. Palma Méndez

Héctor Ruiz Ríos

Arquitectura Intel 80386

- Características principales:
  - **Bus de datos y direcciones de 32 bits**
  - **Registros internos de 32 bits**
  - Puede controlar 4 Gb. Memoria RAM (2^{32})
  - Frecuencias de reloj de 16 a 40 Mhz.
  - Capacidad de gestionar 64 Tb. mem. virtual
  - Cambio dinámico tamaño bus (6, 16, 32 bits)
  - **Soporta multitarea**

Arquitectura Intel 80386

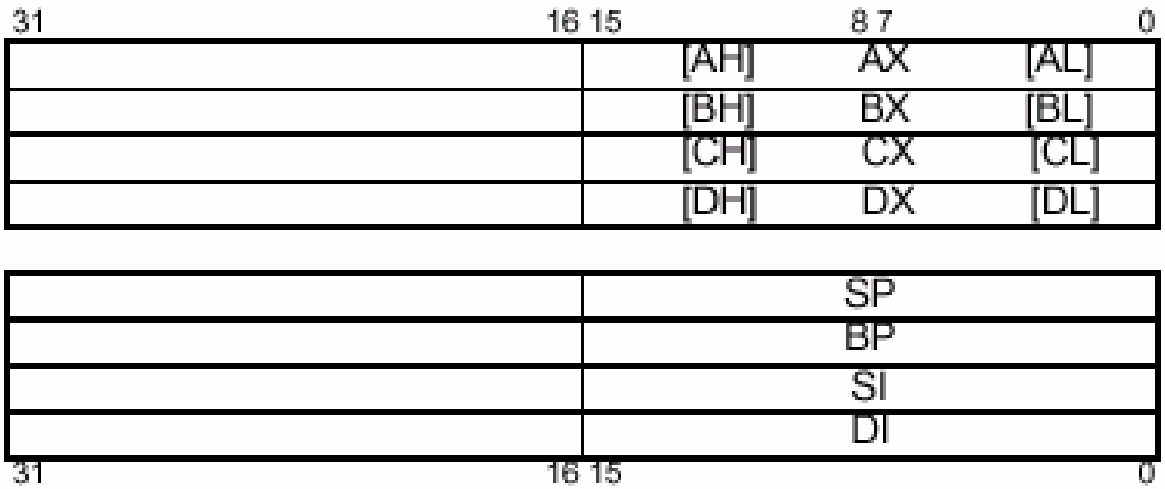
- Características principales:
  - Admite 3 modos de operación:
    - **Modo real:** como el 8086
    - **Modo protegido:** soporta las nuevas prestaciones
    - **Modo virtual-86:** ejecutar programas igual que en el 8086 pero incorporando las principales características del entorno protegido.

Arquitectura Intel 80386

- Registros internos:
  - Propósito general (8)
    - *EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI*
  - Registro de puntero de instrucciones (*EIP*)
  - Registro de flags (*EFLAGS*)
  - Registros de segmento (6)
    - *CS, DS, SS, ES, FS, GS*
    - ***IDTR, GDTR, LDTR***

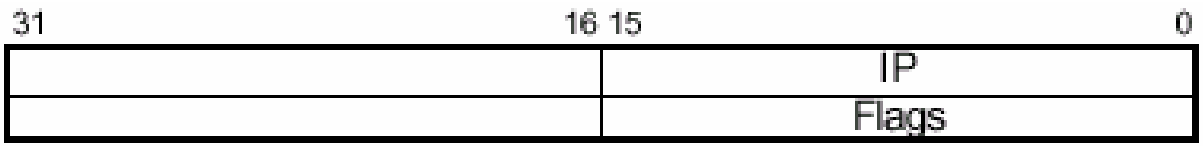
Arquitectura Intel 80386

- Acceso a los registros:



EAX Acumulador
EBX Base
ECX Contador
EDX Datos

ESP Puntero a pila (stack pointer)
EBP Puntero a base (base pointer)
ESI Índice fuente (source index)
EDI Índice destino (destin. index)



EIP Puntero de instrucción
EFlags Registro de flags

Arquitectura Intel 80386

- Direccionamiento en *modo real*
  - Una dirección lógica está formada por:
    - **Selector:** contenido del registro de segmento. Indica dónde comienza el segmento.
    - **Desplazamiento:** valor sumado a la base del segmento (selector)

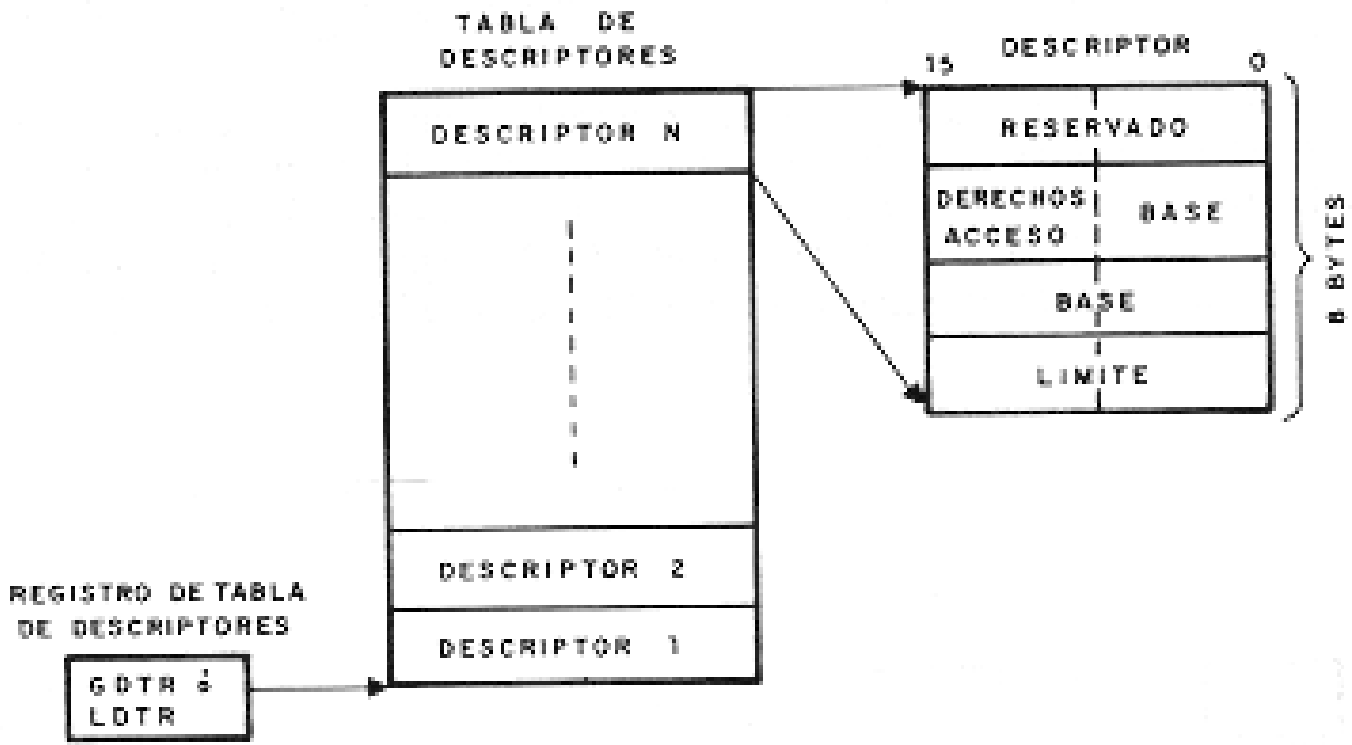
SELECTOR**:**DESPLZAMIENTO

Arquitectura Intel 80386

- Direccionamiento en *modo protegido*
  - Un segmento se caracteriza por:
    - **Base:** dónde comienza el segmento
    - **Límite:** determina el tamaño utilizable por el programador. Expresado en *desplazamiento*.
    - **Atributos:**
      - Tipo de segmento (lectura, escritura...)
      - Nivel de privilegio
      - Indicadores relativos a memoria virtual...

Arquitectura Intel 80386

- Direccionamiento en *modo protegido*



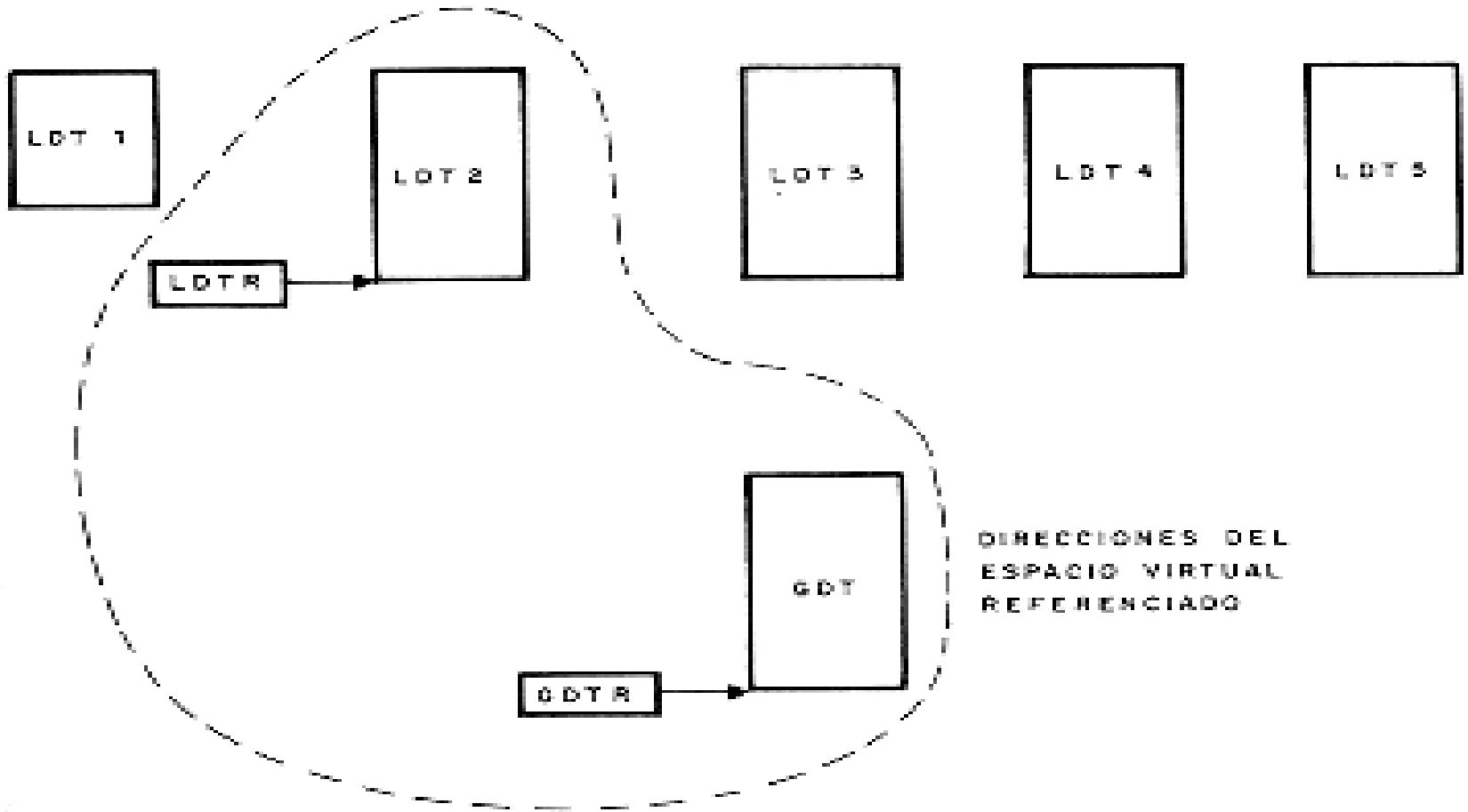
Arquitectura Intel 80386

- Tablas de descriptores
  - Se usan en *modo protegido*
  - Contienen referencias precisas para los segmentos a usar
  - **GDT:** define cada segmento del área global
  - **LDT:** define los segmentos del área local

Arquitectura Intel 80386

- Tablas de descriptores
  - En todo momento se usa la *GDT* (común para todos los procesos) y una *LDTi* (específica para cada proceso *i)*.
  - ***GDTR*** y ***LDTR*** son los registros que las referencian
  - **Una conmutación de tarea implica un cambio de *LDT* a través del registro *LDTR***

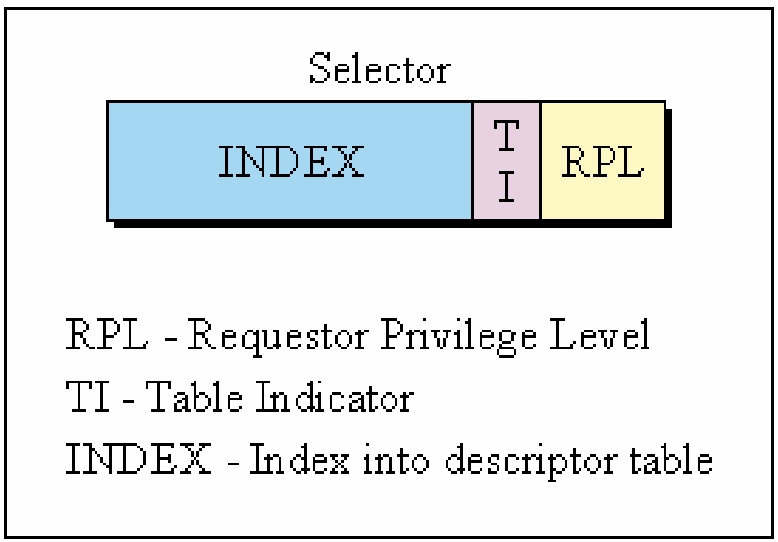
Arquitectura Intel 80386

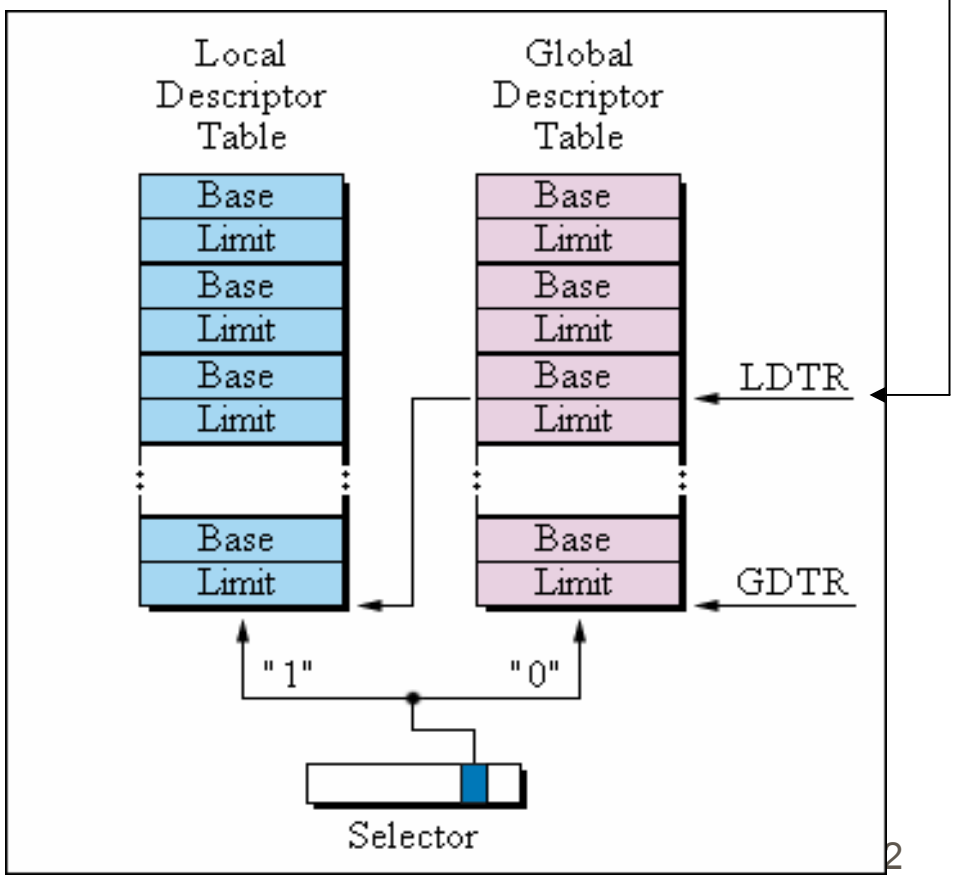


Arquitectura Intel 80386

¡OJO! - Referencia a través de GDT

Formato del selector





Arquitectura Intel 80386

- MULTITAREA
  - El procesador 386 soporta multitarea
  - Al pasar de una tarea a otra debe guardarse el *contexto* (44 bytes) que viene determinado por:
    - Estados de los registros
    - Selector de la LDTi
    - Punteros de pila diversos niveles privilegio

Arquitectura Intel 80386

- Esto se guarda en el segmento ***TSS*** (en la *GDT*), referido por el registro ***TR***.
- ***TR*** actúa como un selector de la GDT, seleccionando uno de sus descriptores que se corresponden con el segmento ***TSS***.
- Ese descriptor poseerá la base y el límite del *TSS*.

Arquitectura Intel 80386

- Conmutación de tarea
  - No hay instrucciones específicas para conmutar tarea
  - Se usan JMP y CALL
  - Se compone de varias etapas
    - Instrucción de conmutación, y salvar en el TSS el contexto
    - Cargar en TR el nuevo valor. Se direcciona el nuevo TSS, y se carga su contenido (contexto) en la CPU.
    - Se reanuda el procesamiento de la nueva tarea

mpx386.s

Este archivo contiene un pedazo de código para inicializar las estructuras de control del MINIX.

En concreto esta es la versión para x386 (32 bits). La versión de 16 bits se realiza en el archivo mpx88.s

La selección de uno u otro se hace en el archivo mpx.s

mpx386.s

- El fichero mpx386.s realiza dos funciones diferenciadas:
  - Inicialización del sistema
  - Manejo de interrupciones,
    - Hardware
    - Software
    - Excepciones

Inicialización del sistema

- Una vez que el proceso de autoarranque ha cargado el S.O. En memoria se transfiere el control a la etiqueta MINIX (línea 6051 en mpx386.s).
- En este punto la CPU ya se encuentra en modo 32 bits (modo protegido).

```
6048    !*===============================
6049    !*                MINIX
6050    !*===============================
6051  MINIX:
6052      jmp     over_flags
6053    .data2    CLICK_SHIFT
6054  flags:
6055    .data2    0x002D
6056
6057
6058    nop
6059  over_flags:
```

Inicialización del sistema

- En la primera instrucción (línea 6052) existe un salto sobre unos cuantos bytes de datos que corresponde a flags del monitor, con las que identificamos varias de las características más importantes del núcleo (por ejemplo, que trabaja en 32 bits).
- SE ESTÁN GUARDANDO DATOS EN EL SEGMENTO DE TEXTO!!! El segmento de datos no está inicializado

```
6048    !*================================
6049    !*                MINIX
6050    !*================================
6051  MINIX:
6052      jmp     over_flags
6053    .data2    CLICK_SHIFT
6054  flags:
6055    .data2    0x002D
6056
6057
6058    nop
6059  over_flags:
6060
```

Inicialización del sistema

- De la línea 6061 a la 6071 se prepara un soporte de pila que ofrezca un entorno apropiado para el código compilado en C.

```
6061  ! Set up a C stack frame on the monitor stack.  (The monitor
6062  ! right.  The ss descriptor still references the monitor
6063              movzx    esp, sp
6064              push     ebp
6065              mov      ebp, esp
6066              push     esi
6067              push     edi
6068              cmp      4(ebp), 0
6069              jz       noret
6070              inc      (_mon_return)
6071  noret:      mov      (_mon_sp), esp
6072
```

Inicialización del sistema

- De la 6073 a la 6100 se encarga de obtener la dirección de la GDT del monitor y la del kernel y transfiere los datos de una a otra. Después carga los parámetros de arranque que necesita el entorno C

```
6073 ! Copiar tabla de descriptores global del mon. en el esp. de dir. del
     ! kernel
6074  ! y conmutar a ella. Asi, prot_init() puede actualizarla con efecto
     ! inmediato
6075
6076      sgdt     (_gdt+GDT_SELECTOR)
6077      mov      esi, (_gdt+GDT_SELECTOR+2) !GDT del monitor
6078      mov      ebx, _gdt                  !GDT del kernel
6079      mov      ecx, 8*8
6080   copygdt:
6081    eseg       movb     al, (esi)
6082               movb     (ebx), al
6083      inc      esi
6084      inc      ebx
6085      loop     copygdt
```

Inicialización del sistema

```
6086  mov     eax, (_gdt+DS_SELECTOR+2)
6087  and     eax, 0x00FFFFFF
6088  add     eax, _gdt
6089  mov     (_gdt+GDT_SELECTOR+2), eax  ;Se fija la base de GDT
6090  lgdt    (_gdt+GDT_SELECTOR)   ;Se carga en el procesador
6091
6092   ! Localizar paráms. de arranque, preparar registros de
      ! segmento de kernel y pila
6093  mov     ebx, 8(ebp)
6094  mov     edx, 12(ebp)
6095  mov     ax, ds
6096  mov     es, ax
6097  mov     fs, ax
6098  mov     gs, ax
6099  mov     ss, ax
6100  mov     esp, k_stktop
```

Inicialización del sistema

- A partir de aquí se llama (línea 6109) a la función ***cstart*** (escrita en C), donde se realizan ciertos procesos como la inicialización de la Tabla de Descriptores Global, GDT, la Tabla de Descriptores de Interrupción, IDT, y los mecanismos de protección de memoria.

```
6102  ! Call C startup code to set up a proper environment to run main().
6103     push    edx
6104     push    ebx
6105     push    SS_SELECTOR
6106     push    MON_CS_SELECTOR
6107     push    DS_SELECTOR
6108     push    CS_SELECTOR
6109     call    _cstart  ! cstart(cs, ds, mcs, mds, parmoff, parmlen)
6110     add     esp, 6*4
6111
6112  ! Reload gdtr, idtr and the segment registers to global
6113  ! descriptor table set up by prot_init().
6114
6115     lgdt    (_gdt+GDT_SELECTOR)
6116     lidt    (_gdt+IDT_SELECTOR)
```

Inicialización del sistema

- Al retornar de ***cstart*** se hacen efectivas las tablas de descriptores recién creadas mediante las instrucciones LGDT y LIDT de las líneas 6115 y 6116.

```
6102  ! Call C startup code to set up a proper environment to run main().
6103     push     edx
6104     push     ebx
6105     push     SS_SELECTOR
6106     push     MON_CS_SELECTOR
6107     push     DS_SELECTOR
6108     push     CS_SELECTOR
6109     call     _cstart  ! cstart(cs, ds, mcs, mds, parmoff, parmlen)
6110     add      esp, 6*4
6111
6112  ! Reload gdtr, idtr and the segment registers to global
6113  ! descriptor table set up by prot_init().
6114
6115     lgdt     (_gdt+GDT_SELECTOR)
6116     lidt     (_gdt+IDT_SELECTOR)
```

Inicialización del sistema

- La instrucción ***jmpf CS_SELECTOR:csinit***, que a simple vista tan sólo salta a la siguiente línea, es una parte importante del proceso de inicialización ya que fuerza el uso de las tablas inicializadas.

```
6114
6115                lgdt    (_gdt+GDT_SELECTOR)
6116                lidt    (_gdt+IDT_SELECTOR)
6117
6118                jmpf    CS_SELECTOR:csinit
6119    csinit:
6120        o16     mov     ax, DS_SELECTOR
6121                mov     ds, ax
6122                mov     es, ax
6123                mov     fs, ax
6124                mov     gs, ax
6125                mov     ss, ax
```

Inicialización del sistema

- Después todos los registros de segmentos normales cambian al segmento de datos y se carga la TSS actual, ya que no cambia.
- MINIX termina con un salto (no una llamada) en la línea 6131 al punto de entrada ***main*** del Kernel (en main.c) y del cual nunca retornará. En este punto ha terminado el código de inicialización del mpx386.s

```
6118                    jmpf    CS_SELECTOR:csinit
6119    csinit:
6120       o16          mov     ax, DS_SELECTOR
6121                    mov     ds, ax
6122                    mov     es, ax
6123                    mov     fs, ax
6124                    mov     gs, ax
6125                    mov     ss, ax
6126       o16          mov     ax, TSS_SELECTOR  !no other TSS is used
6127                    ltr     ax
6128                    push    0 ! poner banderas en un estado estable
6129                    popf      ! elim. tarea anid. y habil. int.
6130
6131                    jmp     _main     ! main()
```

Manejo de Interrupciones MINIX

⌘Tipos en el 80386

- El 80386 soporta tres tipos de fuentes de interrupción:

  1) **Interrupción Hardware:** Producida por dispositivos externos como el reloj, el teclado, etc...

  2) **Interrupción Software**: Producida por la ejecución de instrucciones específicas (*INT* o *INTO).*

  3) **Excepción**: Producida por el propio procesador al encontrar una situación anormal, producido y detectado en el desarrollo del programa en el curso de la ejecución.

Manejo de Interrupciones MINIX

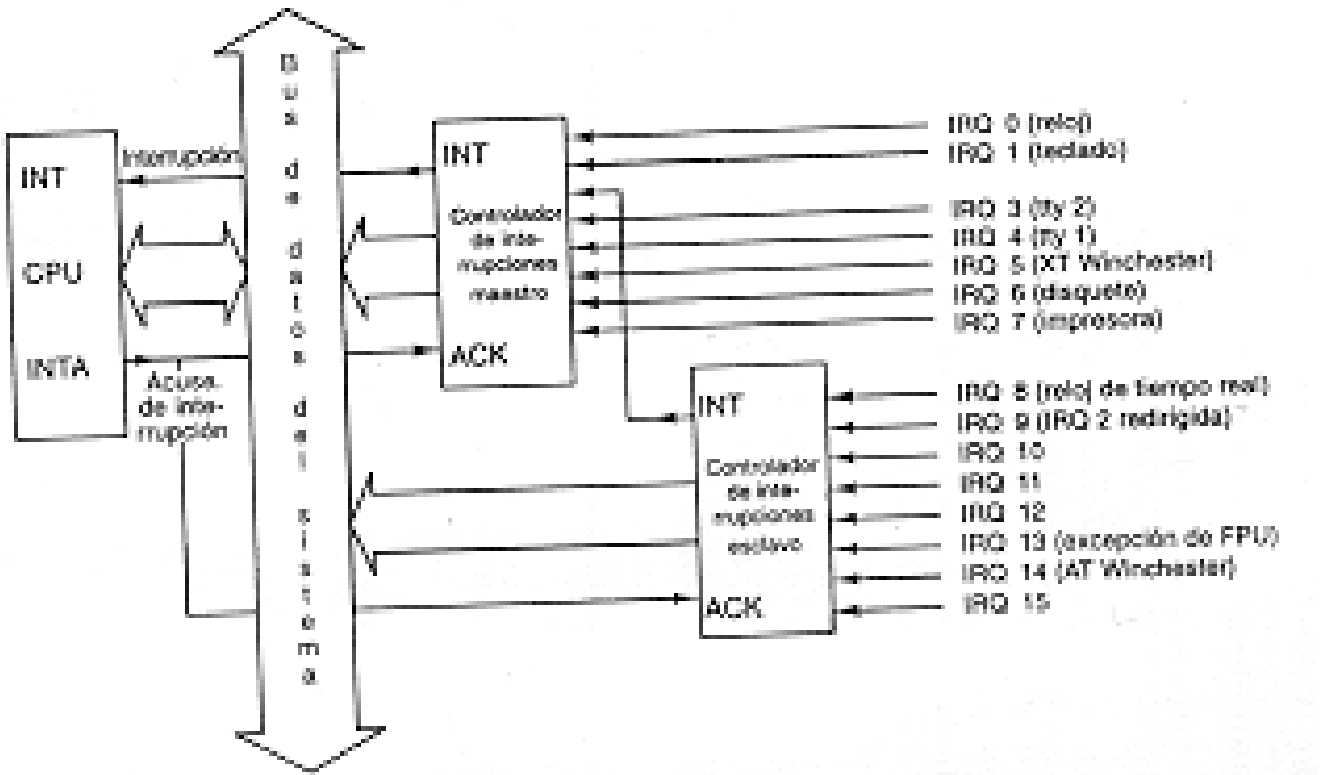
- Interrupciones y excepciones
  - Provocan la detención del programa actual.
  - Primero se salva en pila la dirección de retorno.
  - Al finalizar el servicio de la interrupción, reinicia otro proceso que no tiene porque ser el que la causó.

Manejo de Interrupciones MINIX

- Interrupciones Hardware
  - En el Intel 386 las interrupciones Hardware se controlan mediante dos chips, uno maestro y otro esclavo.
  - El chip esclavo tiene su salida irq 9 conectada a la entrada irq 2 del maestro, que es el que emite la interrupción al procesador por medio de la señal INT(Fig.1).
  - *MINIX* usa macros para manejarlas porque una macro es más rápida que una subrutina.

Manejo de Interrupciones MINIX

- Fig. 1: Interrupciones Hardware en 80386



Manejo de Interrupciones MINIX

- Tratamiento en modo protegido

  • El 80386 dispone de una tabla que contiene 256 ***descriptores de puertas***. Cada descriptor consta de 8 bytes, dando lugar a una tabla de descriptores (puertas) de 2 KB.

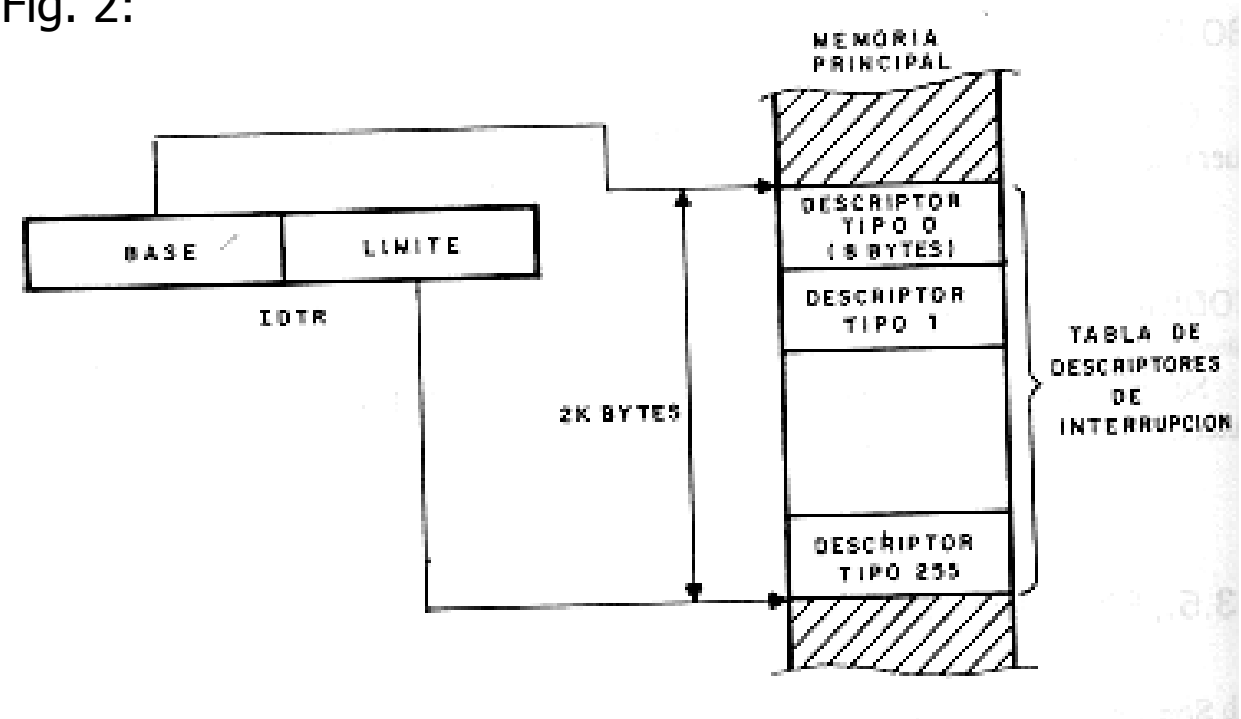
  • Esta **Tabla de Interrupciones** o **IDT**, se localiza en memoria física empleando el registro **IDTR** del procesador (como ocurre con la GDT y el GDTR) que es cargado con una instrucción LIDT durante la inicialización del S.O.(Fig. 2).

  • La tabla de MINIX tiene 56 elementos, de los cuáles sólo se usan realmente 35. Los demás, están reservados para ser utilizados en posibles futuras mejoras del MINIX.

Manejo de Interrupciones MINIX

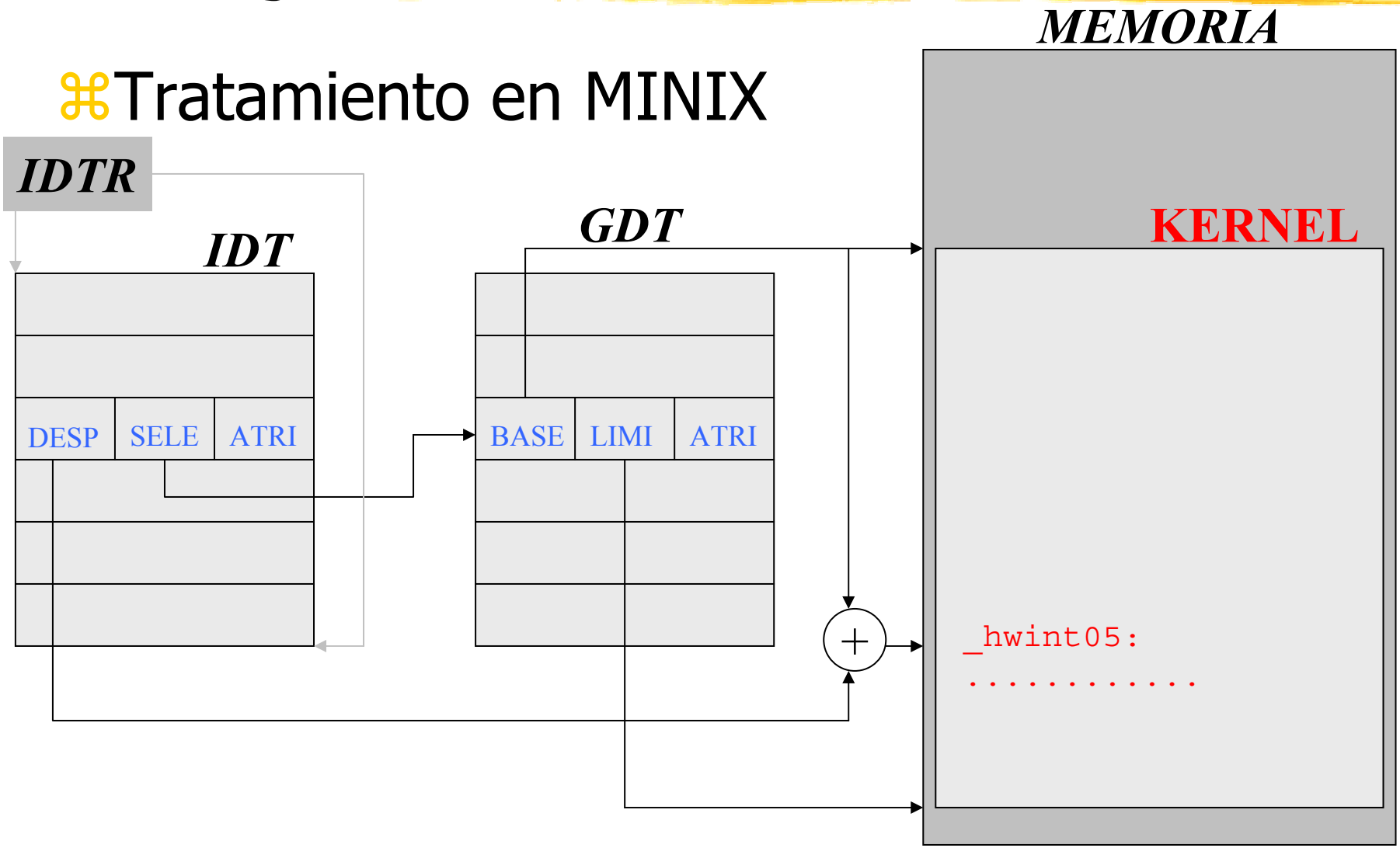
Tratamiento en modo protegido

- **IDTR** se compone de *base física* y *límite* de la tabla de descriptores.
  Fig. 2:



Manejo de Interrupciones MINIX

- Tratamiento en MINIX



Manejo de Interrupciones MINIX

- **Interrupciones Hardware en MINIX**
  - MINIX tiene para cada una de las interrupciones un punto de entrada, tanto para las del dispositivo master (irq0-irq7) como para las del slave (irq8-irq15).
  - Cada una de estas entradas usa una macro para atender al dispositivo que interrumpe.

Manejo de Interrupciones MINIX

HWINT_MASTER

- Entradas a las interrupciones hardware 00-07

```
6164 _hwint00:         ! Interrupt routine for irq 0 (the clock).
6165 hwint_master(0)
6166
6167 .align   16
6168 _hwint01:         ! Interrupt routine for irq 1 (keyboard)
6169 hwint_master(1)
6170
6171 .align   16
6172 _hwint02:         ! Interrupt routine for irq 2 (cascade!)
6173 hwint_master(2)
6174
6175 .align   16
6176 _hwint03:         ! Interrupt routine for irq 3 (second serial)
6177 hwint_master(3)
6178
6179 .align   16
```

Manejo de Interrupciones MINIX

HWINT_MASTER

```
6180 _hwint04:          ! Interrupt routine for irq 4 (first serial)
6181  hwint_master(4)
6182
6183  .align   16
6184 _hwint05:          ! Interrupt routine for irq 5 (XT winchester)
6185  hwint_master(5)
6186
6187  .align   16
6188 _hwint06:          ! Interrupt routine for irq 6 (floppy)
6189  hwint_master(6)
6190
6191  .align   16
6192 _hwint07:          ! Interrupt routine for irq 7 (printer)
6193  hwint_master(7)
```

Manejo de Interrupciones MINIX

HWINT_MASTER

- Una vez detectado el número de la interrupción se llama a la subrutina save().

```
06139   !*===========================================================
06140   !*                                    hwint00 - 07
06141   !*===========================================================
06142   ! Note this is a macro, it looks like a subroutine.
06143   #define hwint_master(irq)          \
06144           call    save                    /* save interrupted process state
06145           inb     INT_CTLMASK
06146           orb     al, [1<<irq]
06147           outb    INT_CTLMASK             /* disable the irq
06148           movb    al, ENABLE
06149           outb    INT_CTL                 /* reenable master 8259
06150           sti                             /*enable interrupts
06151           push    irq                     /* irq
06152           call    (_irq_table + 4*irq)    /* eax = (*irq_table[irq])(irq)
06153           pop     ecx
06154           cli                             /* disable interrupts
```

Manejo de Interrupciones MINIX

HWINT_MASTER

- Desactiva la irq de la interrupción que se está tratando. No se pueden producir interrupciones del mismo tipo anidadas.

```
06139   !*===========================================================
06140   !*                                hwint00 - 07
06141   !*===========================================================
06142   ! Note this is a macro, it looks like a subroutine.
06143   #define hwint_master(irq)          \
06144           call    save                    /* save interrupted process state
06145           inb     INT_CTLMASK
06146           orb     al, [1<<irq]
06147           outb    INT_CTLMASK             /* disable the irq
06148           movb    al, ENABLE
06149           outb    INT_CTL                 /* reenable master 8259
06150           sti                             /*enable interrupts
06151           push    irq                     /* irq
06152           call    (_irq_table + 4*irq)    /* eax = (*irq_table[irq])(irq)
06153           pop     ecx
06154           cli                             /* disable interrupts
```

Manejo de Interrupciones MINIX

HWINT_MASTER

- Se activa el controlador de interrupciones master ya que se desactiva automáticamente cuando se produce una interrupción

```
06139   !*===========================================================
06140   !*                                hwint00 - 07
06141   !*===========================================================
06142   ! Note this is a macro, it looks like a subroutine.
06143   #define hwint_master(irq)           \
06144           call    save                    /* save interrupted process state
06145           inb     INT_CTLMASK
06146           orb     al, [1<<irq]
06147           outb    INT_CTLMASK             /* disable the irq
06148           movb    al, ENABLE
06149           outb    INT_CTL                 /* reenable master 8259
06150           sti                             /*enable interrupts
06151           push    irq                     /* irq
06152           call    (_irq_table + 4*irq)    /* eax = (*irq_table[irq])(irq)
06153           pop     ecx
06154           cli                             /* disable interrupts
06155           test    eax, eax                /* need to reenable irq?
```

Manejo de Interrupciones MINIX

HWINT_MASTER

- Se activan las interrupciones, pues estas se desactivan automáticamente cuando se produce una interrupción.
- Se permiten interrupciones anidadas.

```
06139   !*===========================================================
06140   !*                                hwint00 - 07
06141   !*===========================================================
06142   ! Note this is a macro, it looks like a subroutine.
06143   #define hwint_master(irq)           \
06144           call    save                    /* save interrupted process state
06145           inb     INT_CTLMASK
06146           orb     al, [1<<irq]
06147           outb    INT_CTLMASK             /* disable the irq
06148           movb    al, ENABLE
06149           outb    INT_CTL                 /* reenable master 8259
06150           sti                             /*enable interrupts
06151           push    irq                     /* irq
06152           call    (_irq_table + 4*irq)    /* eax = (*irq_table[irq])(irq)
06153           pop     ecx
06154           cli                             /* disable interrupts
06155           test    eax, eax                /* need to reenable irq?
```

Manejo de Interrupciones MINIX

HWINT_MASTER

- Llama al código específico para el dispositivo utilizando el número de la interrupción.

```
06139   !*===========================================================
06140   !*                                hwint00 - 07
06141   !*===========================================================
06142   ! Note this is a macro, it looks like a subroutine.
06143   #define hwint_master(irq)          \
06144           call    save                    /* save interrupted process state
06145           inb     INT_CTLMASK
06146           orb     al, [1<<irq]
06147           outb    INT_CTLMASK             /* disable the irq
06148           movb    al, ENABLE
06149           outb    INT_CTL                 /* reenable master 8259
06150           sti                             /*enable interrupts
06151           push    irq                     /* irq
06152           call    (_irq_table + 4*irq)    /* eax = (*irq_table[irq])(irq)
06153           pop     ecx
06154           cli                             /* disable interrupts
06155           test    eax, eax                /* need to reenable irq?
06156           jz      0f
```

Manejo de Interrupciones MINIX

HWINT_MASTER

- Se desactivan las interrupciones.

```
06139   !*===========================================================
06140   !*                                hwint00 - 07
06141   !*===========================================================
06142   ! Note this is a macro, it looks like a subroutine.
06143   #define hwint_master(irq)          \
06144           call    save                    /* save interrupted process state
06145           inb     INT_CTLMASK
06146           orb     al, [1<<irq]
06147           outb    INT_CTLMASK             /* disable the irq
06148           movb    al, ENABLE
06149           outb    INT_CTL                 /* reenable master 8259
06150           sti                             /*enable interrupts
06151           push    irq                     /* irq
06152           call    (_irq_table + 4*irq) /* eax = (*irq_table[irq])(irq)
06153           pop     ecx
06154           cli                             /* disable interrupts
06155           test    eax, eax                /* need to reenable irq?
06156           jz      0f
06157           inb     INT_CTLMASK
06158           andb    al, ~[1<<irq]
06159           outb    INT_CTLMASK             /* enable the irq
06160   0:      ret                             /* restart (another) process
```

Manejo de Interrupciones MINIX

HWINT_MASTER

- Se permite otra vez que se produzcan interrupciones de este tipo.

```
06143   #define hwint_master(irq)           \
06144           call    save                    /* save interrupted process state
06145           inb     INT_CTLMASK
06146           orb     al, [1<<irq]
06147           outb    INT_CTLMASK             /* disable the irq
06148           movb    al, ENABLE
06149           outb    INT_CTL                 /* reenable master 8259
06150           sti                             /*enable interrupts
06151           push    irq                     /* irq
06152           call    (_irq_table + 4*irq) /* eax = (*irq_table[irq])(irq)
06153           pop     ecx
06154           cli                             /* disable interrupts
06155           test    eax, eax                /* need to reenable irq?
06156           jz      0f
06157           inb     INT_CTLMASK
06158           andb    al, ~[1<<irq]
06159           outb    INT_CTLMASK             /* enable the irq
06160   0:      ret                             /* restart (another) process
```

Manejo de Interrupciones MINIX

HWINT_MASTER

- Retornar. Se salta al punto de entrada _restart (línea 6322) o al punto restart1 (línea 6337)

```
06143   #define hwint_master(irq)           \
06144           call    save                    /* save interrupted process state
06145           inb     INT_CTLMASK
06146           orb     al, [1<<irq]
06147           outb    INT_CTLMASK             /* disable the irq
06148           movb    al, ENABLE
06149           outb    INT_CTL                 /* reenable master 8259
06150           sti                             /*enable interrupts
06151           push    irq                     /* irq
06152           call    (_irq_table + 4*irq) /* eax = (*irq_table[irq])(irq)
06153           pop     ecx
06154           cli                             /* disable interrupts
06155           test    eax, eax                /* need to reenable irq?
06156           jz      0f
06157           inb     INT_CTLMASK
06158           andb    al, ~[1<<irq]
06159           outb    INT_CTLMASK             /* enable the irq
06160   0:      ret                             /* restart (another) process
```

Manejo de Interrupciones MINIX

HWINT_SLAVE

- El hwint_slave es similar a hwint_master, con la diferencia de que debe volver a habilitar los controladores maestro y esclavo, ya que ambos quedan inhabilitados cuando el esclavo recibe una interrupción.
- **Entradas a las interrupciones hardware 08-15**

```
_hwint08:                ! Interrupt routine for irq 8 (realtime clock)
    hwint_slave(8)

    .align        16
_hwint09:                ! Interrupt routine for irq 9 (irq 2 redirected)
    hwint_slave(9)

    .align        16
_hwint10:                ! Interrupt routine for irq 10
    hwint_slave(10)
```

Manejo de Interrupciones MINIX

HWINT_SLAVE

```
    .align        16
_hwint11:                ! Interrupt routine for irq 11
    hwint_slave(11)

    .align        16
_hwint12:                ! Interrupt routine for irq 12
    hwint_slave(12)

    .align        16
_hwint13:                ! Interrupt routine for irq 13 (FPU exception)
    hwint_slave(13)

    .align        16
_hwint14:                ! Interrupt routine for irq 14 (AT winchester)
    hwint_slave(14)

    .align        16
_hwint15:                ! Interrupt routine for irq 15
    hwint_slave(15)
```

Manejo de Interrupciones MINIX

HWINT_SLAVE

- Habilita tanto al controlador maestro como al controlador esclavo.

```
06195   !*===========================================================================
06196   !*                                hwint08 - 15
06197   !*===========================================================================
06198   ! Note this is a macro, it looks like a subroutine.
06199   #define hwint_slave(irq)        \
06200           call    save                    /* save interrupted process state
06201           inb     INT2_CTLMASK
06202           orb     al, [1<<[irq-8]]
06203           outb    INT2_CTLMASK            /* disable the irq
06204           movb    al, ENABLE
06205           outb    INT_CTL                 /* reenable master 8259
06206           jmp     .+2                     /* delay
06207           outb    INT2_CTL        /* reenable slave 8259
06208           sti                             /* enable interrupts
06209           push    irq                     /* irq
06210           call    (_irq_table + 4*irq)    * eax = (*irq_table[irq])(irq)
06211           pop     ecx
```

Manejo de Interrupciones MINIX

HWINT_SLAVE

```
06212             cli                                   /* disable interrupts
06213             test    eax, eax                      /* need to reenable irq?
06214             jz      0f
06215             inb     INT2_CTLMASK
06216             andb    al, ~[1<<[irq-8]]
06217             outb    INT2_CTLMASK                  /* enable the irq
06218  0:         ret                                   /* restart (another) process
```

Manejo de Interrupciones MINIX

SAVE

- La principal función de SAVE es salvar el contexto del proceso que fue interrumpido.
- Utiliza la variable _k_reenter para contar y determinar el nivel de anidación de las interrupciones.
- Se sale del SAVE con un salto para evitar conflictos con las pilas.

Manejo de Interrupciones MINIX

SAVE

- Guarda el contexto del proceso interrumpido en la pila provista por la CPU.

```
06253   !*===========================================================================
06254   !*                                    save
06255   !*===========================================================================
  ...
06261   save:
06262                   cld                             ! set direction flag to a known value
06263                   pushad                          ! save "general" registers
06264           o16     push    ds                      ! save ds
06265           o16     push    es                      ! save es
06266           o16     push    fs                      ! save fs
06267           o16     push    gs                      ! save gs
06268                   mov     dx, ss                  ! ss is kernel data segment
06269                   mov     ds, dx                  ! load rest of kernel segments
06270                   mov     es, dx                  ! kernel does not use fs, gs
06271                   mov     eax, esp                ! prepare to return
06272                   incb    (_k_reenter)            ! from -1 if not reentering
06273                   jnz     set_restart1            ! stack is already kernel stack
06274                   mov     esp, k_stktop
06275                   push    _restart                ! build return address for int handler
06276                   xor     ebp, ebp                ! for stacktrace
06277                   jmp     RETADR-P_STACKBASE(eax)
```

Manejo de Interrupciones MINIX

SAVE

- Uso de la variable _k_reenter para controlar el nivel de anidamiento de las interrupciones.

```
06253   !*===========================================================================
06254   !*                                    save
06255   !*===========================================================================
  ...
06261   save:
06262                   cld                             ! set direction flag to a known value
06263                   pushad                          ! save "general" registers
06264       o16         push    ds                      ! save ds
06265       o16         push    es                      ! save es
06266       o16         push    fs                      ! save fs
06267       o16         push    gs                      ! save gs
06268                   mov     dx, ss                  ! ss is kernel data segment
06269                   mov     ds, dx                  ! load rest of kernel segments
06270                   mov     es, dx                  ! kernel does not use fs, gs
06271                   mov     eax, esp                ! prepare to return
06272                   incb    (_k_reenter)            ! from -1 if not reentering
06273                   jnz     set_restart1            ! stack is already kernel stack
06274                   mov     esp, k_stktop
06275                   push    _restart                ! build return address for int handler
06276                   xor     ebp, ebp                ! for stacktrace
06277                   jmp     RETADR-P_STACKBASE(eax)
```

Manejo de Interrupciones MINIX

SAVE

- Si hay anidamiento mete en la pila la dirección de _restart, si no mete la dirección de _restart1.
- Esta dirección será usada por la rutina que llamo a save()

```
06268                   mov     dx, ss                  ! ss is kernel data segment
06269                   mov     ds, dx                  ! load rest of kernel segments
06270                   mov     es, dx                  ! kernel does not use fs, gs
06271                   mov     eax, esp                ! prepare to return
06272                   incb    (_k_reenter)            ! from -1 if not reentering
06273                   jnz     set_restart1            ! stack is already kernel stack
06274                   mov     esp, k_stktop
06275                   push    _restart                ! build return address for int handler
06276                   xor     ebp, ebp                ! for stacktrace
06277                   jmp     RETADR-P_STACKBASE(eax)
06278
06279           .align  4
06280   set_restart1:
06281                   push    restart1
06282                   jmp     RETADR-P_STACKBASE(eax)
```

Manejo de Interrupciones MINIX

SAVE

- Se retorna con un salto ya que la pila ha sido modificada.

```
06261        save:
06262                    cld                                  ! set direction flag to a known value
06263                    pushad                               ! save "general" registers
06264          o16       push        ds                       ! save ds
06265          o16       push        es                       ! save es
06266          o16       push        fs                       ! save fs
06267          o16       push        gs                       ! save gs
06268                    mov         dx, ss                   ! ss is kernel data segment
06269                    mov         ds, dx                   ! load rest of kernel segments
06270                    mov         es, dx                   ! kernel does not use fs, gs
06271                    mov         eax, esp                 ! prepare to return
06272                    incb        (_k_reenter)             ! from -1 if not reentering
06273                    jnz         set_restart1             ! stack is already kernel stack
06274                    mov         esp, k_stktop
06275                    push        _restart                  build return address for int handler
06276                    xor         ebp, ebp       ! for stacktrace
06277                    jmp         RETADR-P_STACKBASE(eax)
06278
06279                    .align      4
06280        set_restart1:
06281                    push        restart1
06282                    jmp         RETADR-P_STACKBASE(eax)
```

Manejo de Interrupciones MINIX

- **Interrupciones Software en MINIX**
  - En MINIX las interrupciones software se tratan en s_call.
  - Se diferencian de las hardware en que el índice a la IDT se toma de la propia instrucción y no del controlador.

Manejo de Interrupciones MINIX

S_CALL

- Guarda el contexto del proceso interrumpido en la pila provista por la CPU.

```
06284 !*===========================================================
06285 !*                              s_call
06286 !*===========================================================
06287         .align      16
06288     _s_call:
06289 _p_s_call:
              ...                                  ! inline save
06307                     sti                      ! allow SWITCHER to be interrupted
06308                                              ! now set up parameters for sys_call()
06309                     push ebx                 ! pointer to user message
06310                     push eax                 ! src/dest
06311                     push ecx                 ! SEND/RECEIVE/BOTH
06312                     call_sys_call            ! sys_call(function, src_dest, m_ptr)
06313                                              ! caller is now explicitly in proc_ptr
06314                     mov AXREG(esi), eax      ! sys_call MUST PRESERVE si
06315                     cli                      ! disable interrupts
              ! Fall into code to restart proc/task running.
...           ! continua en _restart
```

Manejo de Interrupciones MINIX

S_CALL

- Llama a sys_call (implementada en C) que convierte la interrupción en un mensaje.

```
06284 !*===========================================================
06285 !*                              s_call
06286 !*===========================================================
06287         .align      16
06288     _s_call:
06289 _p_s_call:
              ...                                   ! inline save
06307                     sti                       ! allow SWITCHER to be interrupted
06308                                               ! now set up parameters for sys_call()
06309                     push ebx                  ! pointer to user message
06310                     push eax                  ! src/dest
06311                     push ecx                  ! SEND/RECEIVE/BOTH
06312                     call_sys_call             ! sys_call(function, src_dest, m_ptr)
06313                                               ! caller is now explicitly in proc_ptr
06314                     mov AXREG(esi), eax       ! sys_call MUST PRESERVE si
06315                     cli                       ! disable interrupts
              ! Fall into code to restart proc/task running.
...           ! continua en _restart
```

Manejo de Interrupciones MINIX

RESTART

- Reinicia un proceso, tal vez diferente al que se interrumpió.
- Esta función comprueba si hay interrupciones no atendidas que llegaron mientras se estaba tratando la actual.

Manejo de Interrupciones MINIX

RESTART

- Se selecciona el proceso que va a ser reinicializado (puede ser distinto del que se interrumpió).

```
06319   !*===========================================================================*
06320   !*                                restart                                    *
06321   !*===========================================================================*
06322   _restart:
06329                   cmp     (_held_head), 0         ! do fast test to usually avoid function call
06330                   jz      over_call_unhold
06331                   call    _unhold                 ! this is rare so overhead acceptable
06332   over_call_unhold:
06333                   mov     esp, (_proc_ptr)        ! will assume P_STACKBASE == 0
06334                   lldt    P_LDT_SEL(esp)          ! enable segment descriptors for task
06335                   lea     eax, P_STACKTOP(esp)    ! arrange for next interrupt
06336                   mov     (_tss+TSS3_S_SP0), eax  ! to save state in process table
06337   restart1:
06338                   decb    (_k_reenter)
06339      o16          pop     gs
06340      o16          pop     fs
06341      o16          pop     es
06342      o16          pop     ds
06343                   popad
06344                   add     esp, 4                  ! skip return adr
06345                   iretd                           ! continue process
```

Manejo de Interrupciones MINIX

RESTART

- Se decrementa _k_reenter. Quitamos un nivel de anidamiento.

```
06319         !*===========================================================================
06320         !*                                    restart
06321         !*===========================================================================
06322         _restart:
06329                     cmp     (_held_head), 0         ! do fast test to usually avoid function call
06330                     jz      over_call_unhold
06331                     call    _unhold                 ! this is rare so overhead acceptable
06332         over_call_unhold:
06333                     mov     esp, (_proc_ptr)        ! will assume P_STACKBASE == 0
06334                     lldt    P_LDT_SEL(esp)          ! enable segment descriptors for task
06335                     lea     eax, P_STACKTOP(esp)    ! arrange for next interrupt
06336                     mov     (_tss+TSS3_S_SP0), eax  ! to save state in process table
06337         restart1:
06338                     decb    (_k_reenter)
06339           o16       pop     gs
06340           o16       pop     fs
06341           o16       pop     es
06342           o16       pop     ds
06343                     popad
06344                     add     esp, 4                  ! skip return adr
06345                     iretd                           ! continue process
```

Manejo de Interrupciones MINIX

RESTART

- Reinicia el proceso seleccionado.

```
06319   !*===========================================================================
06320   !*                                    restart
06321   !*===========================================================================
06322   _restart:
06329                   cmp     (_held_head), 0         ! do fast test to usually avoid function call
06330                   jz      over_call_unhold
06331                   call    _unhold                 ! this is rare so overhead acceptable
06332   over_call_unhold:
06333                   mov     esp, (_proc_ptr)        ! will assume P_STACKBASE == 0
06334                   lldt    P_LDT_SEL(esp)          ! enable segment descriptors for task
06335                   lea     eax, P_STACKTOP(esp)    ! arrange for next interrupt
06336                   mov     (_tss+TSS3_S_SP0), eax  ! to save state in process table
06337   restart1:
06338                   decb    (_k_reenter)
06339      o16          pop     gs
06340      o16          pop     fs
06341      o16          pop     es
06342      o16          pop     ds
06343                   popad
06344                   add     esp, 4                  ! skip return adr
06345                   iretd                           ! continue process
```

Manejo de Interrupciones MINIX

- **Excepciones en MINIX**
  - Provocadas por el procesador, por ejemplo en casos de overflow o divisiones por cero.
  - Son tratadas de igual forma que las interrupciones.
  - Se manejan con la rutina ***exception*** (implementada en C).

Manejo de Interrupciones MINIX

EXCEPCIONES

- Lista de las posibles excepciones en MINIX.

```
!*===========================================================================*
!*                                exception handlers                         *
!*===========================================================================*
_divide_error:
        push    DIVIDE_VECTOR
        jmp     exception

_single_step_exception:
        push    DEBUG_VECTOR
        jmp     exception

_nmi:
        push    NMI_VECTOR
        jmp     exception

_breakpoint_exception:
        push    BREAKPOINT_VECTOR
        jmp     exception

_overflow:
        push    OVERFLOW_VECTOR
        jmp     exception
```

Manejo de Interrupciones MINIX

EXCEPCIONES

```
_bounds_check:
        push    BOUNDS_VECTOR
        jmp     exception

_inval_opcode:
        push    INVAL_OP_VECTOR
        jmp     exception

_copr_not_available:
        push    COPROC_NOT_VECTOR
        jmp     exception

_double_fault:
        push    DOUBLE_FAULT_VECTOR
        jmp     errexception

_copr_seg_overrun:
        push    COPROC_SEG_VECTOR
        jmp     exception

_inval_tss:
        push    INVAL_TSS_VECTOR
        jmp     errexception
```

Manejo de Interrupciones MINIX

EXCEPCIONES

```
_segment_not_present:
        push    SEG_NOT_VECTOR
        jmp     errexception

_stack_exception:
        push    STACK_FAULT_VECTOR
        jmp     errexception

_general_protection:
        push    PROTECTION_VECTOR
        jmp     errexception

_page_fault:
        push    PAGE_FAULT_VECTOR
        jmp     errexception

_copr_error:
        push    COPROC_ERR_VECTOR
        jmp     exception
```

Manejo de Interrupciones MINIX

EXCEPCIONES

- Todas estas excepciones saltan a **exception o errexception** dependiendo de si la condición agrega a la pila un código de error o no.

```
!*=============================================================================
!*                                    exception                                *
!*=============================================================================
! This is called for all exceptions which do not push an error code.

        .align  16
exception:
 sseg   mov     (trap_errno), 0          ! clear trap_errno
 sseg   pop     (ex_number)
        jmp     exception1
```

Manejo de Interrupciones MINIX

EXCEPCIONES

```
!*=============================================================================*
!*                                    errexception                             *
!*=============================================================================*
! This is called for all exceptions which push an error code.

        .align  16
errexception:
 sseg   pop     (ex_number)
 sseg   pop     (trap_errno)
exception1:                                     ! Common for all exceptions.
        push    eax                             ! eax is scratch register
        mov     eax, 0+4(esp)                   ! old eip
 sseg   mov     (old_eip), eax
        movzx   eax, 4+4(esp)                   ! old cs
 sseg   mov     (old_cs), eax
        mov     eax, 8+4(esp)                   ! old eflags
 sseg   mov     (old_eflags), eax
        pop     eax
        call    save
        push    (old_eflags)
        push    (old_cs)
        push    (old_eip)
        push    (trap_errno)
```

Manejo de Interrupciones MINIX

EXCEPCIONES

```
push    (ex_number)
call    _exception                  ! (ex_number, trap_errno, old_eip,
                                    !       old_cs, old_eflags)
add     esp, 5*4
cli
ret
```

Preguntas

- ¿Por qué existen dos partes distintas de código para tratar las interrupciones hardware?
- ¿Qué son y qué función desempeñan las tablas GDT, LDT e IDT?
- ¿En el arranque del mpx386.s, por qué no se genera una nueva TSS del kernel?
- ¿Qué función desempeña mpx386?