Sistemas Operativos II

Microkernel
MINIX 2° parte

Universidad Arturo Jauretche
Ingeniería Informática

Docentes:

Coordinador: Ing. Jorge Osio

Profesor: Ing. Eduardo Kunysz

Interrupciones

Manejo de Interrupciones MINIX

El 80386 soporta tres tipos de fuentes de interrupción:

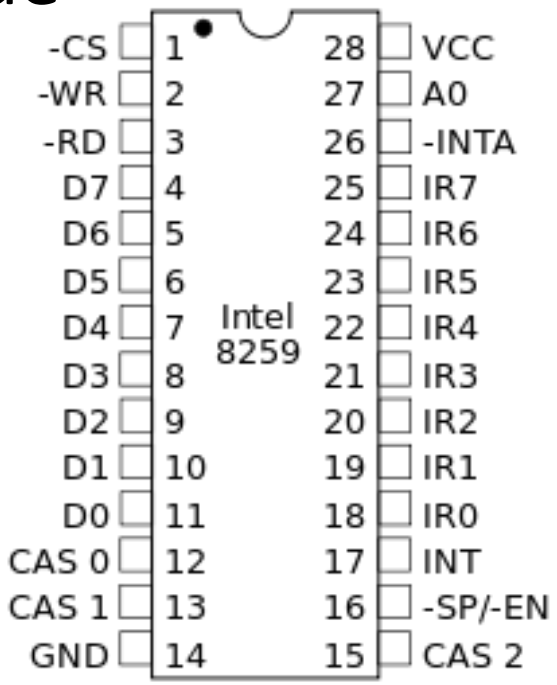
1) **Interrupción Hardware:** Producida por dispositivos externos como el reloj, el teclado, etc...

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

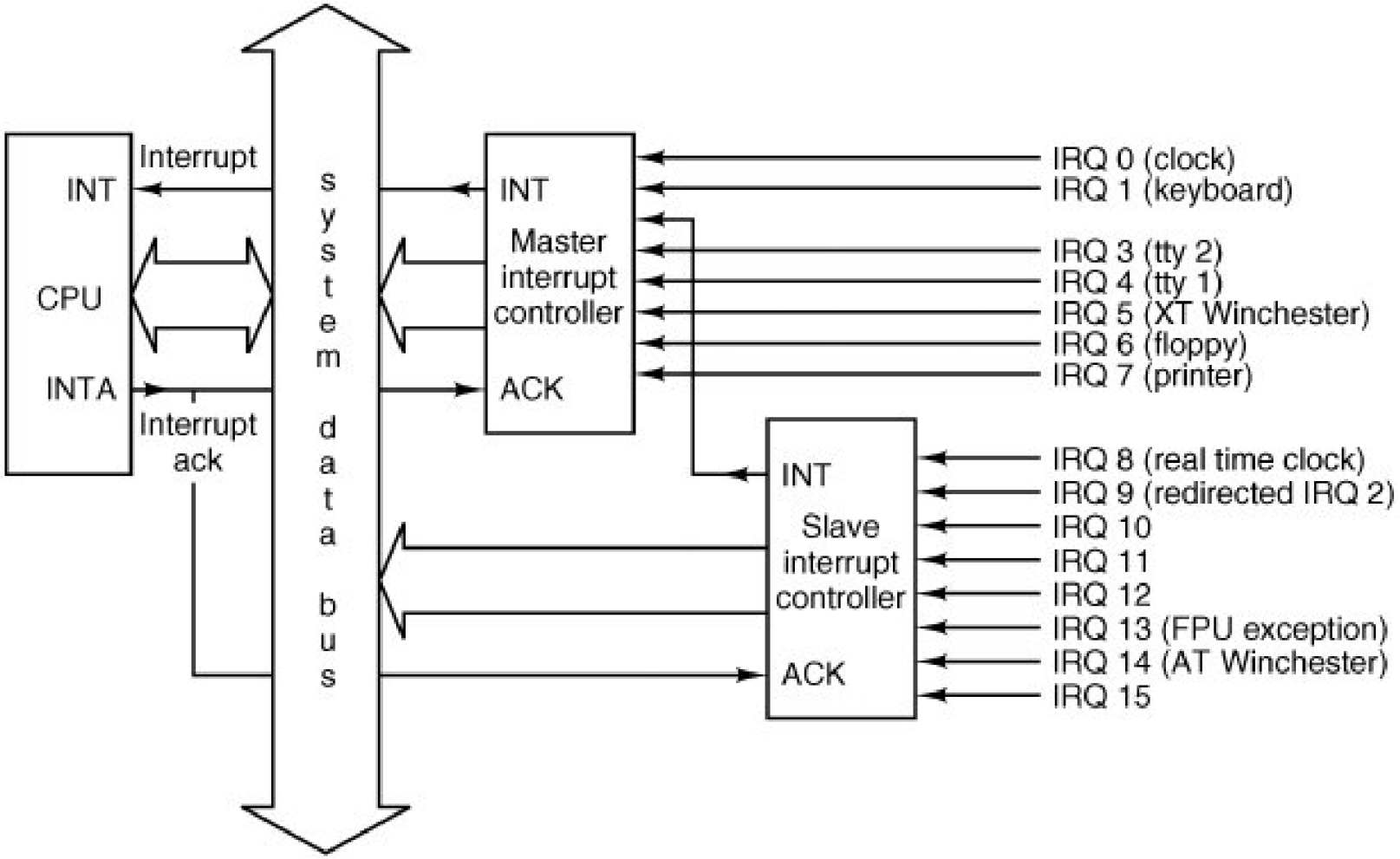
3) **Excepción:** Producida por el propio procesador al encontrar una situación anormal, producido y detectado en el desarrollo del programa en el curso de la ejecución.

Manejo de interrupciones en MINIX

- Las interrupciones generadas por dispositivos de hardware son señales eléctricas.
- Se manejan por un controlador de interrupciones
- Normalmente vienen equipadas con dos de estos chips con una capacidad total de manejo de 15 entradas



Manejo de interrupciones

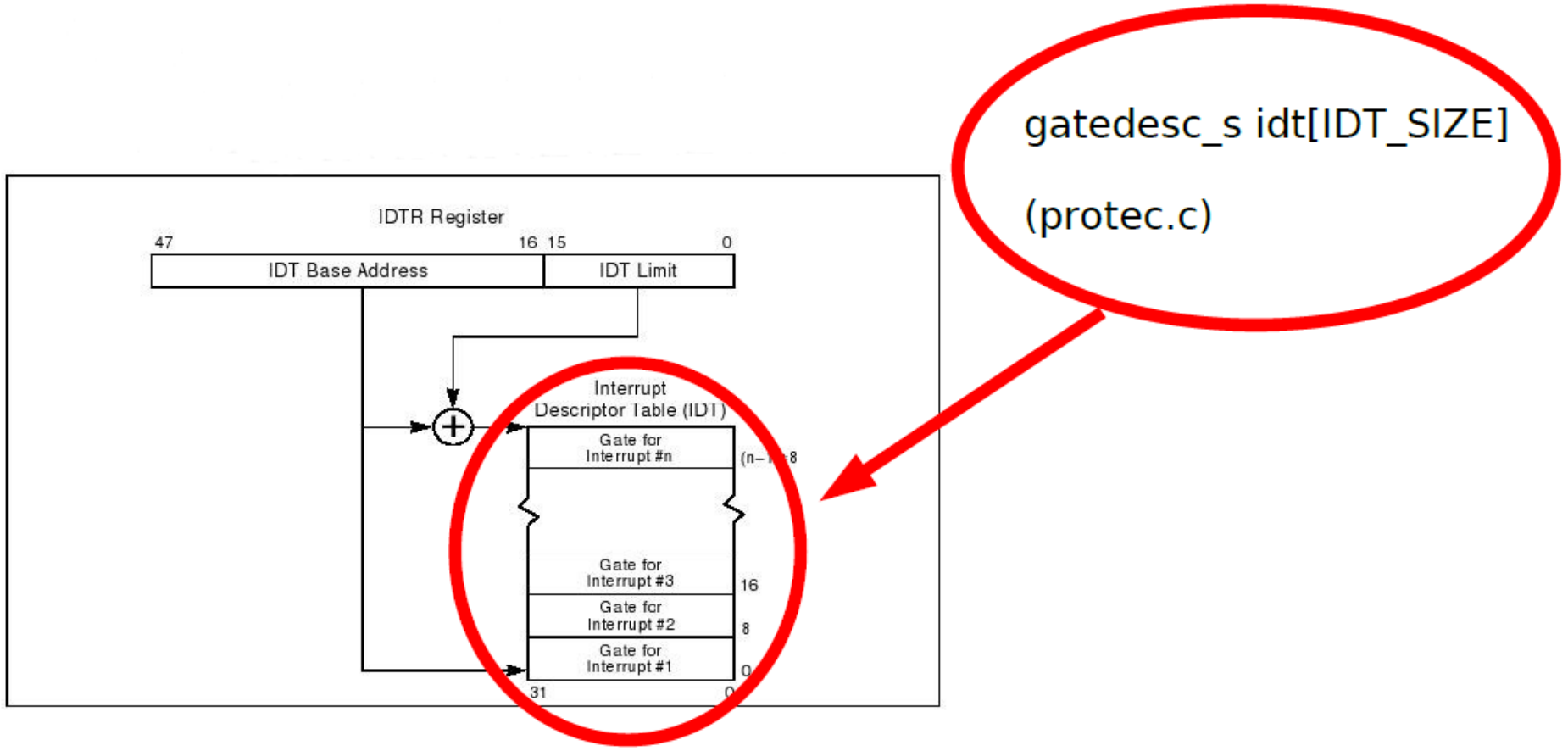


Manejo de interrupciones

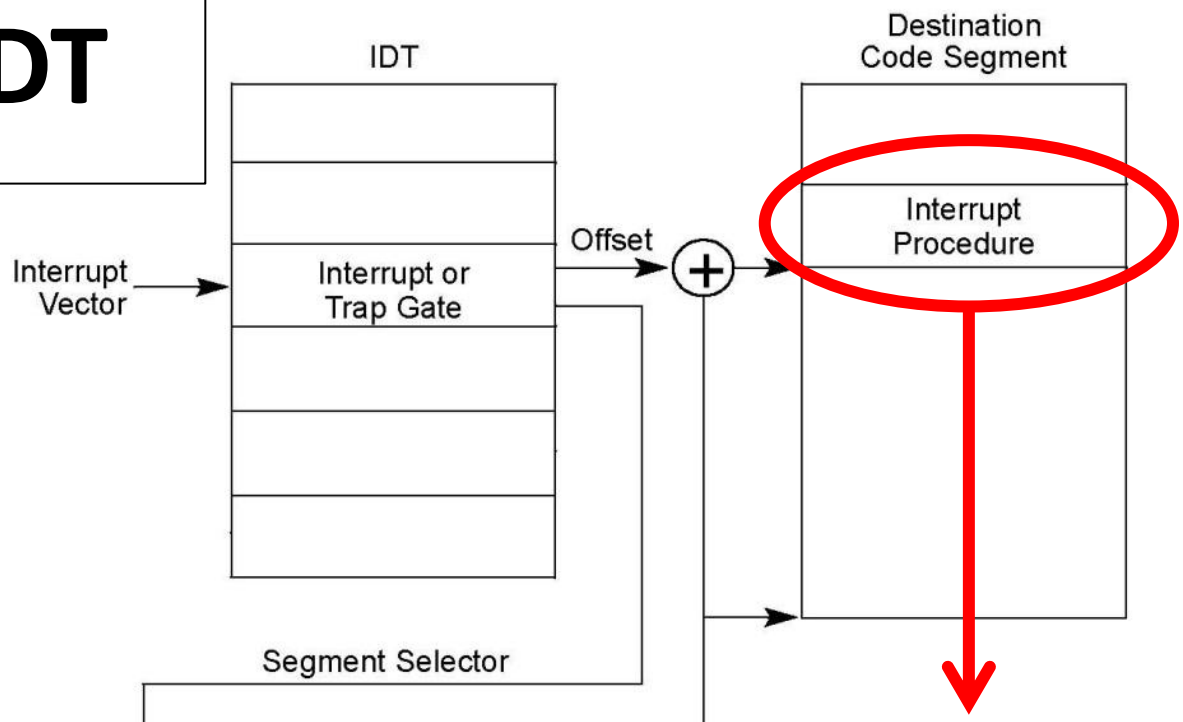
- La conexión **INT** al CPU le dice al procesador que ha ocurrido una interrupción.
- La señal **INTA** del CPU indica al controlador que coloque los datos en el BUS.
- Los chips se programan durante la inicialización por **main.c** en **intr_init**.
- Los datos colocados en el bus son de 8 bits: tabla de 256 elementos. (MINIX tiene 56)

Interrupt Descriptor Table (IDT)

La tabla IDT es inicializada por prot_init (en protect.c) invocado por start.c



Macro IDT



Macro para manejo del 8259 en mpx386.s

GDT: Tabla Descriptor Global

LDT: Tabla Descriptor Local

mpx386.s

- Una pequeña parte del kernel ve realmente las interrupciones. Este código está en mpx386.s
- Hay un punto de entrada para cada interrupción:
  - hwint00 a 07 llamada a master
  - hwint08 a 15 llamada a slave
- Indican cuál dispositivo requiere servicio.

mpx386.s (hwint00 -07)

```
#define hwint_master(irq)        \
        call    save    /* save interrupted process state
        push    (_irq_handlers+4*irq) /* irq_handlers[irq]
        call    _intr_handle
                        /* intr_handle(irq_handlers[irq])
        pop     ecx
        cmp     (_irq_actids+4*irq), 0
                        /* interrupt still active?
        jz      0f
        inb     INT_CTLMASK     /* get current mask
        orb     al, [1<<irq]    /* mask irq
        outb    INT_CTLMASK     /* disable the irq
0:      movb    al, END_OF_INT
        outb    INT_CTL         /* reenable master 8259
        ret                     /* restart (another) process
```

mpx386.s (hwint00 -07)

```
#define hwint_master(irq)
        call    save            /* s                    ate
        push    (_irq_handle                         irq]
        call    _intr_handle
                                /*                     q])
        pop     ecx
        cmp     (_irq_actids+4*irq), 0
                                /* interrupt still active?
        jz      0f
        inb     INT_CTLMASK     /* get current mask
        orb     al, [1<<irq]    /* mask irq
        outb    INT_CTLMASK     /* disable the irq
0:      movb    al, END_OF_INT
        outb    INT_CTL         /* reenable master 8259
        ret                     /* restart (another) process
```

Salva los registros para recuperarlos luego

mpx386.s (hwint00 -07)

```
#define hwint_master(irq)                \
        call    save        /* save interrupted process state
        push    (_irq_handlers+4*irq) /* irq_handlers[irq]
        call    _intr_handle
                            /* intr
        pop     ecx
        cmp     (_irq_actids+4*
                            /* inte
        jz      0f
        inb     INT_CTLMASK
        orb     al, [1<<irq]
        outb    INT_CTLMASK
0:      movb    al, END_OF_INT
        outb    INT_CTL
        ret                         /* restart (another) process
```

Usa el número de interrupción atendida como índice para acceder a una tabla de direcciones de la rutina de bajo nivel (en C) del dispositivo

mpx386.s (hwint00 -07)

```
#define hwint_master(irq)                                    \
        call    save            /* sa                        te
        push    (_irq_handler                                 q]
        call    _intr_handle
                                /* in                         )
        pop     ecx
        cmp     (_irq_actids+
                                /* in
        jz      0f
        inb     INT_CTLMASK         /* get current mask
        orb     al, [1<<irq]        /* mask irq
        outb    INT_CTLMASK         /* disable the irq
0:      movb    al, END_OF_INT
        outb    INT_CTL             /* reenable master 8259
        ret                         /* restart (another) process
```

Libera las interrupciones enmascaradas

mpx386.s (hwint00 -07)

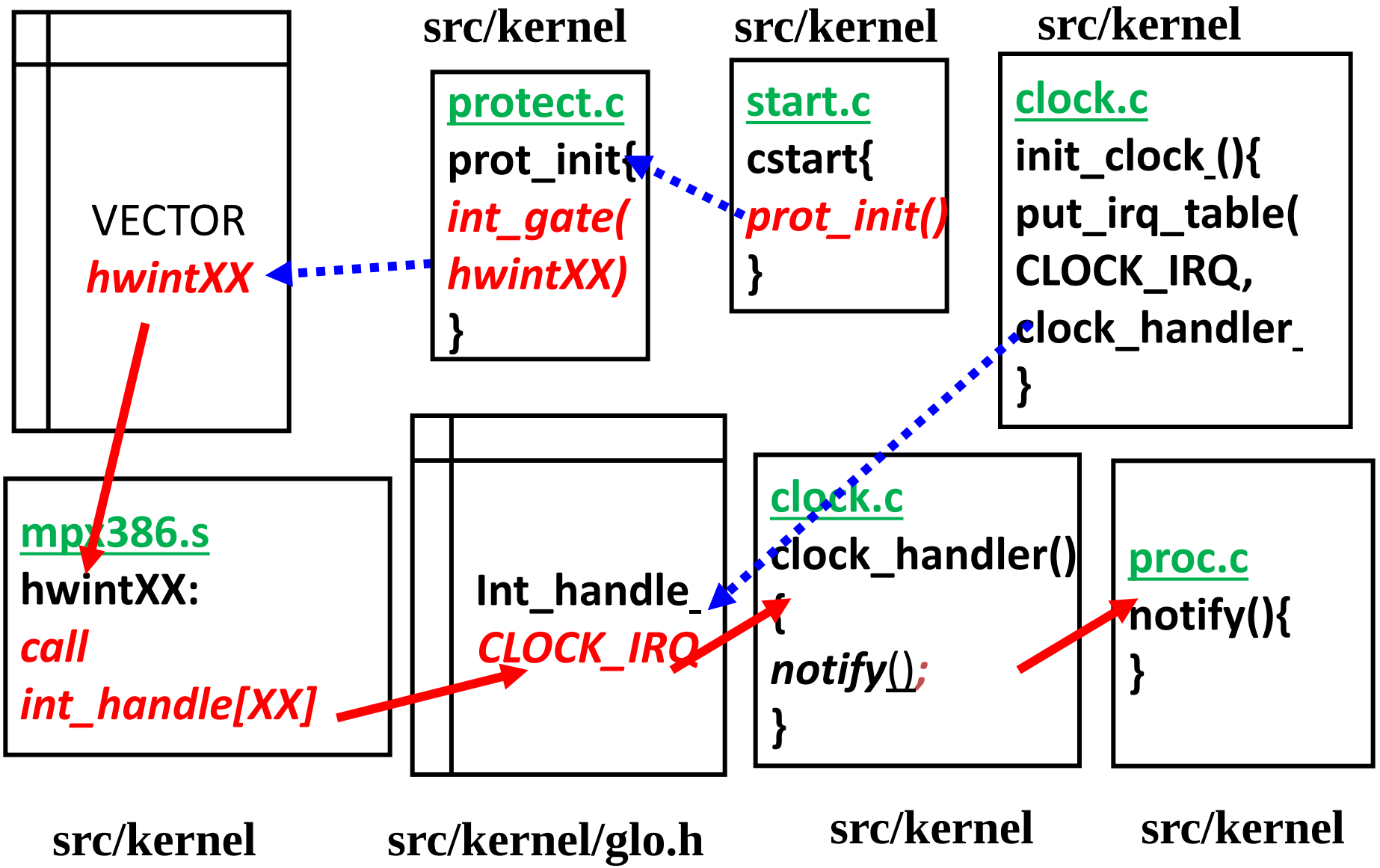
```
#define hwint_master(irq)                \
        call    save    /* save interrupted process state
        push    (_irq_handlers+4*irq) /* irq_handlers[irq]
        call    _intr_handle
                        /* intr_handle(irq_handlers[irq])
        pop     ecx
        cmp     (_irq_actids+4*
                        /* inte
        jz      0f
        inb     INT_CTLMASK
        orb     al, [1<<irq]
        outb    INT_CTLMASK          /* disable the irq
0:      movb    al, END_OF_INT
        outb    INT_CTL              /* reenable master 8259
        ret                          /* restart (another) process
```

Restablece el controlador de interrupciones

mpx386.s (hwint00 -07)

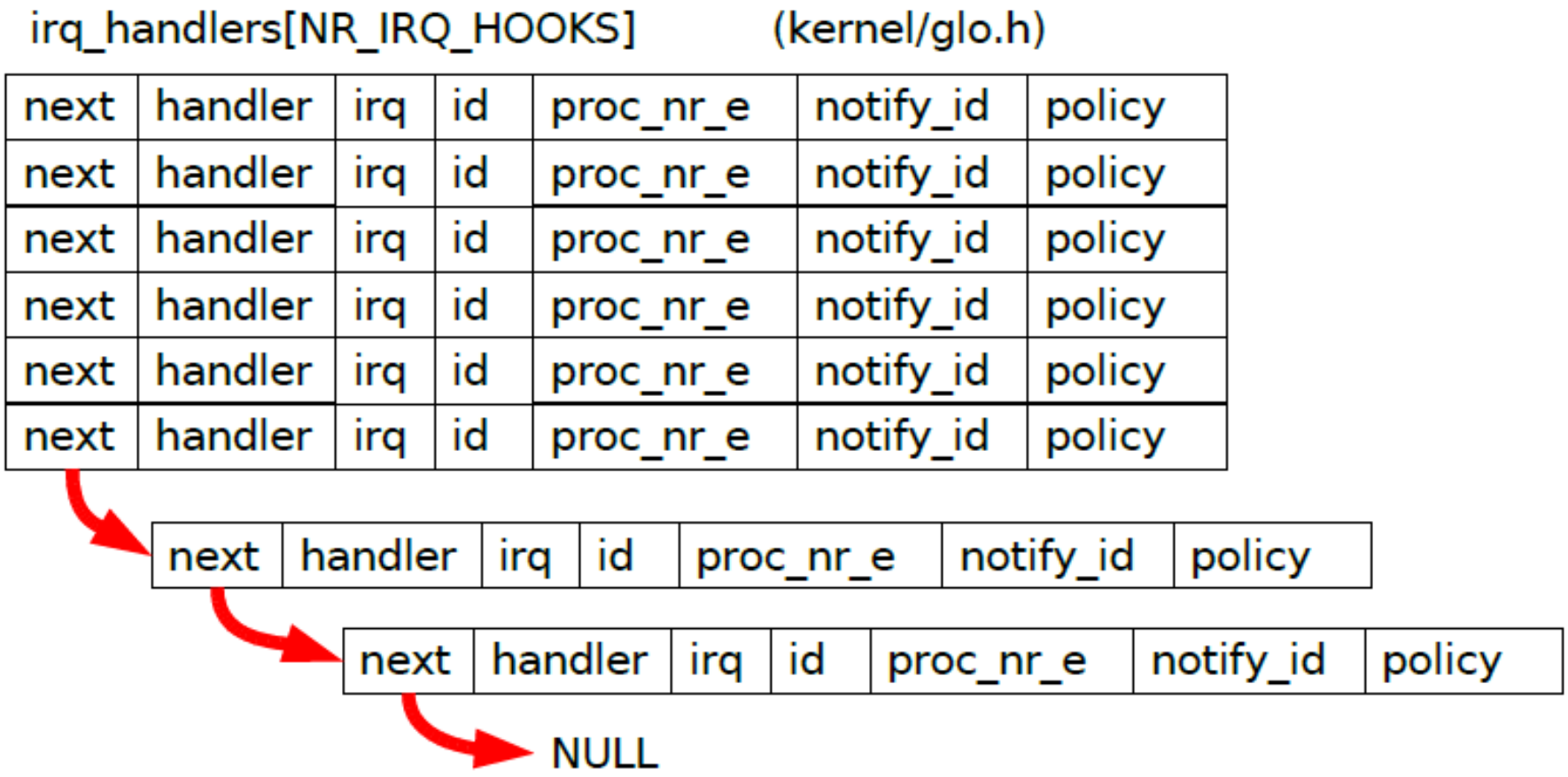
- **intr_handle**: manejador de interrupciones genérico (kernel/i8259.c)
- **irq_actids**: es distinto de cero si el manejador no ha terminado el tratamiento. Debe enmascararse la interrupción para evitar una reentrada de la misma.
- La instrucción "**ret**" realmente nos lleva a _restart, como veremos seguidamente.

Atendiendo Interrupciones



Irq_hook

Estructura de irq_hook_t definido en kernel/type.h

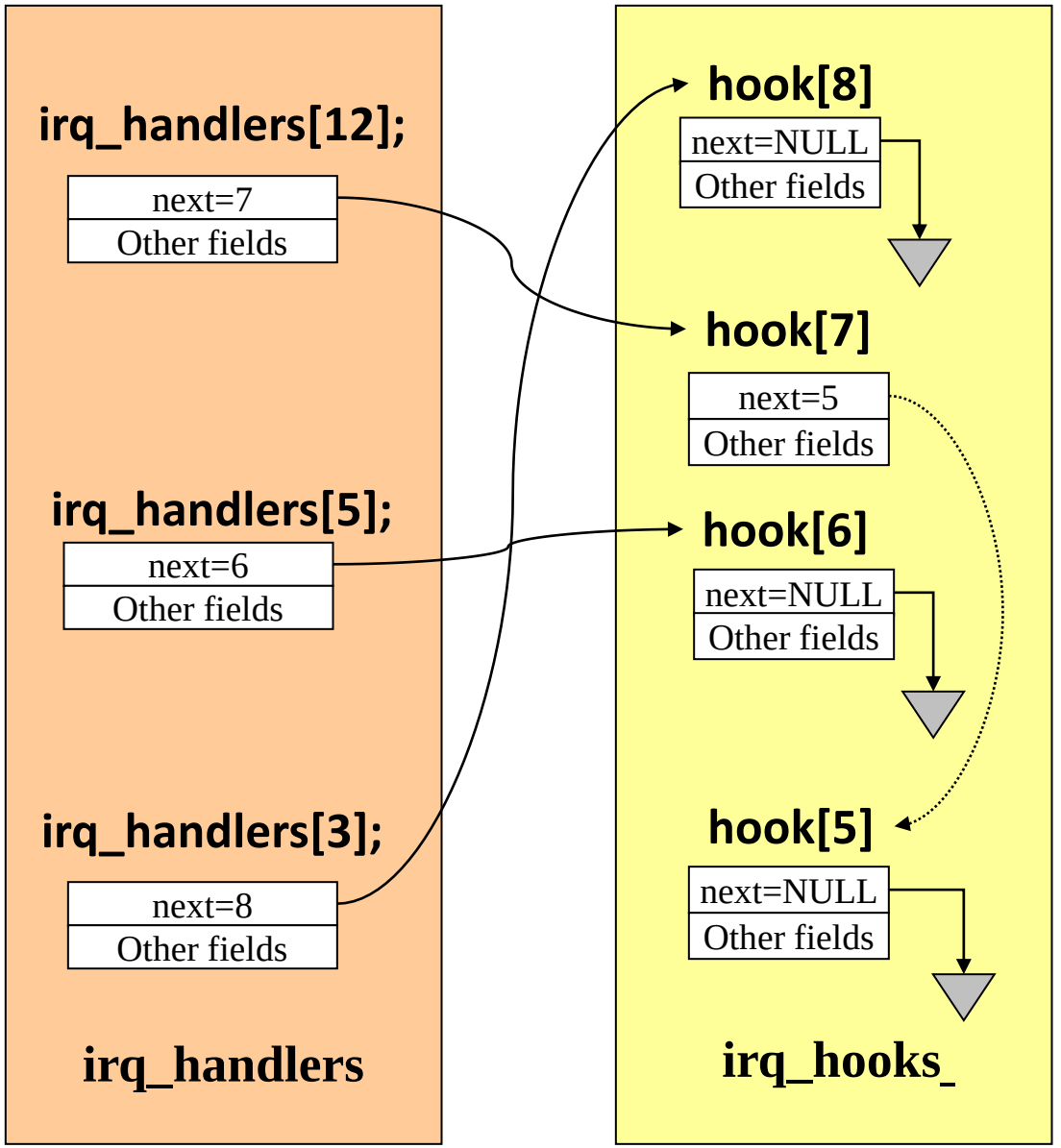


Handle

- **intr_handle** (i8259.c) ejecuta todos los manejadores de los dispositivos encadenados a la misma IRQ. Para cada manejador ejecutado se verifica si ha terminado sus operaciones, en cuyo caso se borra el correspondiente bit de actividad.

Nota: El manejador de interrupción de reloj es el único incorporado en el código del núcleo de MINIX.

Setear un Handle



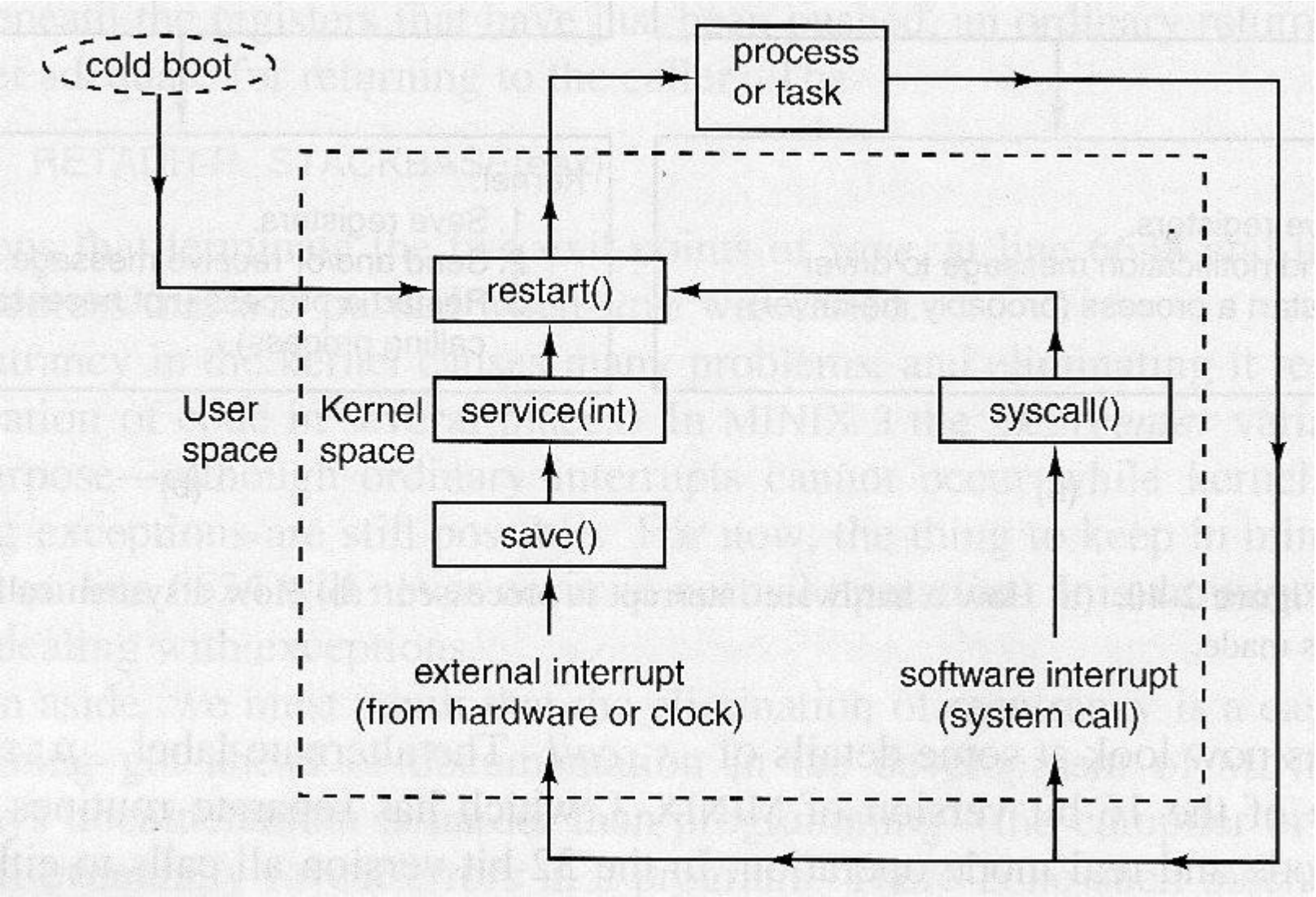
Función SAVE

- Guarda el **contexto** del procesador al momento de producirse la interrupción y lo **intercambia por** el contexto de **kernel** para iniciar los procesos de handle de la interrupción. (Cambia la pila de usuario por la de núcleo).
- **Save introduce en la pila la dirección de _restart** para que al retornar de una interrupción (hwint) se active el proceso que corresponda.

Interrupciones por Software

- No se instala el tratamiento genérico hwintxx. (ver inicialización en protect.c)
- Para la interrupción SYS386_VECTOR se instala **_s_call** (mpx386.s) como tratamiento.
- _s_call llama a **sys_call** (proc.c).
- Cuando sys_call termina y el control vuelve a _s_call, no se ejecutar “ret” sino que el flujo de ejecución continua por **_restart**, produciéndose un cambio de contexto (al igual que ocurre con las interrupciones hardware).

Restart


cold boot
process
or task
restart()
User
space
Kernel
space
service(int)
syscall()
save()
external interrupt
(from hardware or clock)
software interrupt
(system call)

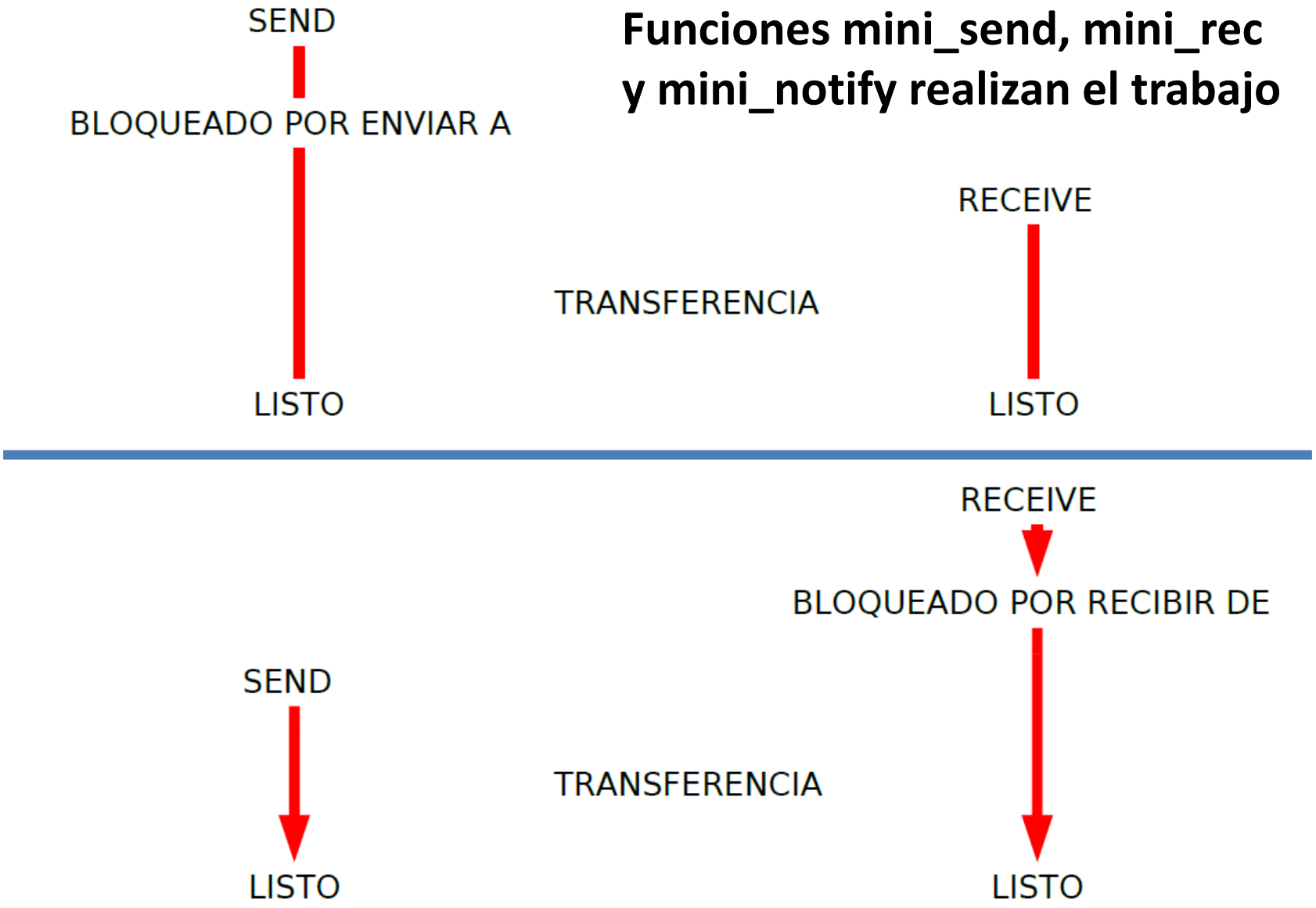
Comunicación entre procesos

Comunicación entre procesos

- El código de alto nivel para el envío de mensajes está en **kernel/proc.c**.
- El núcleo debe **convertir en mensajes las interrupciones** hardware (avisos de dispositivos) y software (llamadas al sistema).
- Como ya hemos visto **_s_call** (mpx386.s) llama a **sys_call** (proc.c) que en definitiva **convierte una interrupción** software **en un mensaje**.

Comunicación entre procesos

Funciones mini_send, mini_rec y mini_notify realizan el trabajo



Sys_call - Mini_send

Sys_call (proc.c)

1- Realiza diferentes comprobaciones.

2- Elabora el mensaje (mini-send o mini-receive o mini-notify o ECHO).

Mini_send (proc.c)

IF proceso destino está esperando este mensaje THEN
    Copiar el mensaje al “buffer” del destino
    Desbloquear al destino (enqueue)
ELSE IF destino no bloqueado o no esperando a origen THEN
    Bloquear al proceso origen (dequeue)
    Encolar al proceso origen en la cola de procesos esperando enviar al destino (en descriptor de proceso destino)
END IF

Mini_receive – Mini_Notify

Mini_receive (proc.c)

IF hay un mensaje del proceso origen THEN
    Adquirir el mensaje
    Desbloquear al proceso origen
ELSE
    Bloquear al proceso destino
    END IF

Mini-notify (proc.c)

Muy similar a mini_send. La mayor diferencia es que el origen no se bloquea.

lock_notify, lock_send, lock_enqueue, lock_dequeue hacen prácticamente lo mismo que los servicios correspondientes pero con la garantía de que las interrupciones hardware están inhibidas.

Archivos de encabezamiento comunes

Headers (1)

minix/config.h establece parámetros para “kernel”.

ansi.h determina si el compilador utilizado cumple los requerimientos de C standard.

limits.h define varios tamaños básicos (número de bits en un entero, límites del sistema operativo, etc.)

errno.h contiene los códigos de error que se devuelven a los programas de usuario.

sys/types.h define tipos de datos para usar dentro de MINIX.

Headers (2)

unistd.h define constantes y prototipos
string.h define prototipos relacionados con el manejo de strings.
signal.h define los nombres estándares de señales y algunos prototipos relacionados con ellas
fcntl.h define parámetros relacionados con operaciones de control de ficheros.
termios.h define constantes, macros y prototipos de funciones relacionadas con el manejo de terminales de entrada salida.
timers.h suministra el soporte para temporizadores.
stdlib.h stdio.h no son utilizados dentro de MINIX, pero se suministran por ser de uso frecuente entre los que programan en C ***a.out.h*** define el formato de los ficheros ejecutables.
stddef.h define algunas macros frecuentes.

Headers (3)

sys/sigcontext.h define estructuras para preservar y restaurar un estado de operación normal del sistema cuando se activa un manejador de señal.
sys/stat.h contiene declaraciones para las llamadas al sistema *stat* y *fstat* así como los prototipos de las mismas y otros relacionados.
sys/dir.h describe la estructura de directorio en MINIX 3.
sys/wait.h define macros utilizadas por *wait* y *waitpid* suministradas por el manejador de procesos (pm).
sys/select.h suministra definiciones necesarias para *svrctl*.
minix/ioctol.h da soporte a la llamada al sistema *ioctl*, la cual suministra diferentes operaciones para el control de dispositivos.

Headers (4)

Los ficheros de **i*nclude/minix*** son necesarios para construir MINIX **sobre cualquier plataforma**.

Según haya diferencias puntuales en el hardware o según sea la forma en que queramos utilizar MINIX, puede ser necesario modificar ***minix/config.h*** o ***minix/sys_config.h*** (que es incluido por el anterior).

minix/const.h contiene declaraciones que generalmente no es necesario modificar, pero que se utilizan en diferentes lugares. Otros ficheros *const.h* pueden encontrarse en otros lugares en MINIX, pero son de uso más limitado.

Headers (5)

minix/type.h introduce algunas estructuras importantes.
Unidades en las que se mide la memoria: **phys_clicks** utilizado por el kernel para acceder a cualquier elemento de memoria o **vir_clicks** para acceder a la memoria por medio de un esquema segmentado desde fuera del kernel.
La estructura **sigmsg** permite indicar al kernel, cuando se activa una señal, que la siguiente vez que planifique al proceso arranque el manejador de la señal.
La estructura **kinfo** permite indicar la ubicación del kernel a otras partes del sistema. PM la utiliza para actualizar la tabla de procesos.
minix/ipc.h contiene la estructura **message** así como las definiciones de los tipos de mensajes y los prototipos de las funciones que permiten operar con los mismos.

Headers (6)

minix/syslib.h contiene prototipos para permitir que algunos componentes del sistema operativo puedan acceder a los servicios de otros componentes. Son muy importantes las macros para poder realizar operaciones de entrada y salida, **sys_sdevio**. Poder solicitar al kernel que realice estas operaciones es fundamental para ejecutar los manejadores de dispositivos en espacio de usuario como hace MINIX 3.

minix/sysutil.h suministra, entre otros servicios, **printf** (no printf de la librería estándar, que necesita escribir en la salida estándar y por tanto el sistema de ficheros, sino un printf utilizado por componentes del sistema) y **kputc** (utilizado por printf y que se declara en diferentes lugares según las necesidades de diferentes componentes del sistema). El kernel utiliza **kprintf** en lugar de printf.

Headers (6)

Los ficheros de encabezamiento agrupados en **include/ibm** suministran definiciones relacionadas con la familia de computadores IBM-PC.

ibm/portio.h suministra servicios para efectuar operaciones de entrada salida en puertos del PC y utilizando diferentes tipos de datos. También suministra rutinas para habilitar o deshabilitar interrupciones.

ibm/interrupt.h define los puertos de entrada salida y las direcciones de memoria utilizadas por el controlador de interrupciones y BIOS.

ibm/ports.h suministra las direcciones utilizadas por el teclado y el “timer”.

Otros ficheros de encabezamiento, profusamente comentados (según el autor) son *bios.h, memory.h, partition.h, cmos.h, cpu.h, int86.h, diskparm.h*

Bibliografía

- Cap 2, Tanembaum "Sistemas Operativos, Diseño e implementación"