Capítulo 8

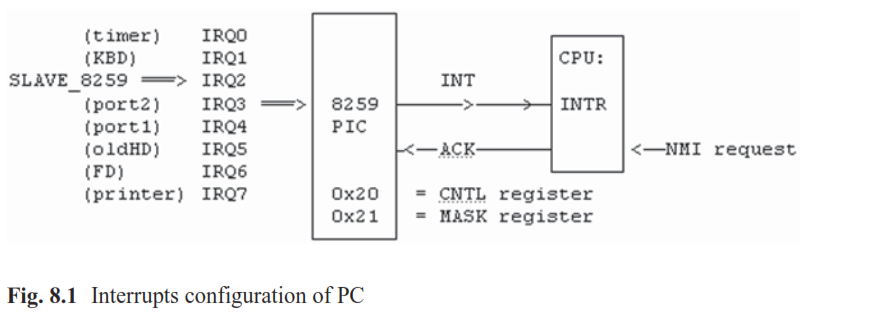
Procesamiento de Interrupciones y Planificación de Procesos

8.1 Interrupciones

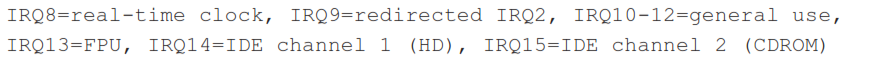
Las interrupciones son señales externas de dispositivos a la CPU, pidiendo servicio de la CPU. La CPU Intel x86 tiene dos líneas de petición de interrupción binarias: NMI, lo cual significa Non-Maskable Interrupt, e INTR, lo cual significa Interrupt Request. Una interrupción está presente si cualquiera de las líneas es activada. La CPU siempre acepta cualquier interrupción NMI, por eso se le llama NMI. Para una petición INTR, la CPU podría enmascarar la petición si el bit I del registro de banderas de la CPU es 0. Una interrupción enmascarada se mantiene pendiente hasta que el flag.I\_bit de la CPU es cambiado a 1. La instrucción CLI limpia el flag.I\_bit y la instrucción STI establece le flag.I\_bit.

8.2 Hardware de Interrupciones

En las PCs basadas en Intel x86, las interrupciones son configuradas como se muestra en la figura 8.1.



El hardware de interrupción consiste de dos Controladores de Interrupción Programables 8259 interconectados (PICs) (Intel 8259A 2014). El primer PIC es el amo (master) y el segundo PIC es el esclavo. Cada PIC tiene 8 entradas, denotadas por IRQ0 a IRQ7. Entre las IRQ’s, IRQ0 tiene la más alta prioridad. El PIC 8259 tiene un codificador de prioridad, el cual se comporta como una caja de pelea de perros (dog-fight box), donde el perro mas grande (top-dog) (la IRQ de prioridad más alta) emerge como el ganador. El PIC envía la IRQ con la prioridad más alta a la entrada INTR de la CPU. La línea de petición INT del PIC esclavo es conectada a la IRQ2 del PIC amo. Las IRQs del PIC esclavo también son conocidas como



Cada PIC 8259 tiene un registro de control de interrupción (INT\_CNTL) y un registro de máscara de interrupción (INT\_MASK).



El registro INT\_MASK es usado para habilitar/deshabilitar líneas IRQ individuales. Una línea IRQn está habilitada si el bit n del registro INT\_MASK es igual a 0; si el bit n es igual a 1 se deshabilita IRQn. El registro INT\_CTL es usado para señalar que se ha terminado un procesamiento de interrupción [End-Of-Interrupt (EOI) processing]. Después de enviar una interrupción IRQn a la CPU, el PIC 8259 automáticamente bloquea todas las IRQs de la misma o menor prioridad. El software manejador de interrupción debe escribir explícitamente EOI al registro INT\_CTL al final del procesamiento de la interrupción. En caso contrario, no podrán ocurrir más interrupciones de la misma o menor prioridad otra vez. La información anterior se resume como sigue.

1 El PIC 8259 ordena las peticiones de interrupción de IRQ0 (la más alta) a IRQ7 (la más baja).

2 Cada IRQn puede ser habilitada/deshabilitada escribiendo al registro INT\_MASK en 0x21 o 0xA1, en el cual bitn=0 habilita IRQn, bitn=1 deshabilita IRQn.

3 Cuando ocurren peticiones de interrupción, el PIC 8259 enruta la IRQn de más alta prioridad a la terminal INTR de la CPU. Mientras tanto, bloquea cualquier petición de interrupción de la misma o menor prioridad, pero enrutará cualquier petición de prioridad más alta a la CPU.

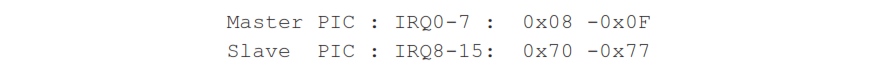
4 Cuando se manda señal a INTR, la CPU no acepta la petición hasta que su bit flag.I es 1. Mientras el bit flag.I es 0, INTR permanecerá señalizada. Lo cual mantiene la petición de interrupción pendiente.

8.3 Procesamiento de interrupción

Cuando una interrupción ocurre, si la flag.I\_bit de la CPU es 1, la CPU acepta la interrupción y comienza a procesar la interrupción al final de la instrucción actual o en un punto interrumpible en una instrucción larga. La secuencia de eventos durante el procesamiento de interrupción es como sigue.

1 La CPU envía una señal INT\_ACK al PIC 8259, pidiendo un vector de interrupción.

2 El PIC 8259 escupe un valor byte, conocido como el vector de interrupción, al bus de datos, permitiendo a la CPU leer el vector. Cuando la PC comienza, los vectores de interrupción de los PICs son inicializados por el BIOS a los valores por defecto



Los vectores de interrupción pueden ser cambiados programando los controladores de interrupción 8259. Dado que es innecesario remapear los vectores IRQ en el modo real de 16 bits, usaremos los valores de vector por defecto.

3 Para PCs en el modo real, el kilobyte de memoria más bajo contiene 256 vectores de interrupción. Cada área de vector contiene un par de [PC,CS], el cual apunta al punto de entrada de un manejador de interrupción. Para manejar una interrupción en IRQn, debe estar instalado un manejador de interrupción para el vector de interrupción antes de que la interrupción ocurra.

4 Después de adquirir el vector n, las acciones de la CPU son idénticas a aquellas que corresponden a ejecutar una instrucción INT n. Primero guarda [flag, CS, PC] en la pila ACTUAL, cambia los bits T e I del registro flag a 0, carga [PC,CS] de la ubicación de vector [4\*n,4\*n+2] y continúa la ejecución desde los nuevos [PC,CS], lo cual es el punto de entrada del manejador de interrupción IRQn.

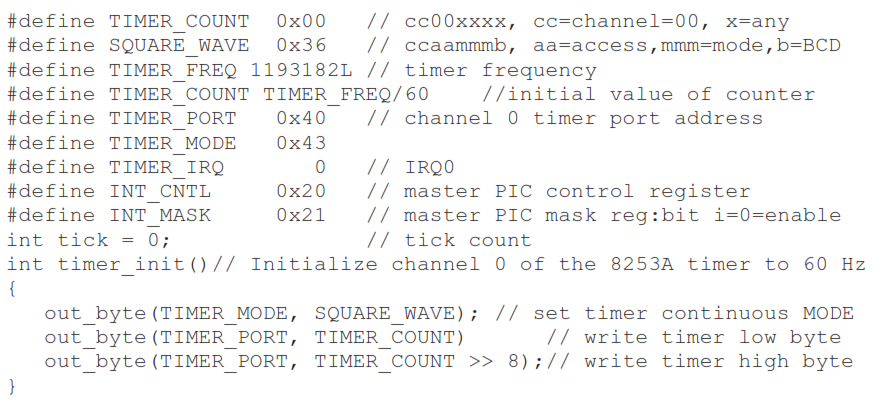
8.4 TIMER y Servicio de TIMER

El timer de la PC (Intel 8253 2010) tiene tres canales independientes. El timer de canal 0 está conectado al PIC amo en la IRQ0. Puede ser programado para generar ondas cuadradas, las cuales son usadas para disparar interrupciones de timer periódicas. En esta sección, incorporaremos el timer de la PC en el kernel MTX para proporcionar funciones de servicio basadas en timer.

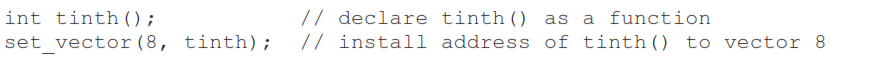
8.4.1 Operación del Timer de la PC

(1) El timer del canal 0 de la PC tiene una frecuencia base de 1193182 Hz. Puede ser programado para generar 60 interrupciones por segundo. El intervalo de tiempo entre interrupciones de timer es llamado un tick, como el tick de un reloj. Este interrumpe en IRQ0 (vector 8).

(2) Inicialización de timer: El segmento de código mostrado abajo inicializa el timer.

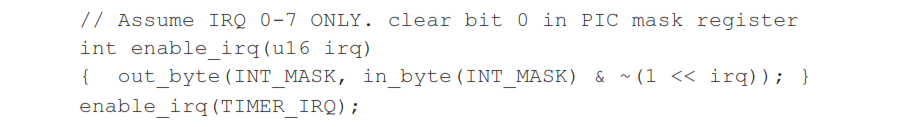


(3) Instalar un manejador de timer al vector 8 y habilitar la interrupción de timer: Suponga que \_tinth es el punto de entrada del manejador de interrupción de timer en código ensamblador. Primero, instalar el manejador de interrupción de timer por



lo cual escribe [\_tinth,0x1000] a [4\*8,4\*8+2], similar a instalar int80h para las syscalls.

Entonces habilitar las interrupciones de timer con



Una vez inicializado y habilitado, el timer interrumpirá en IRQ0 una vez cada 1/60 s. Si el sistema aún no está listo para manejar interrupciones, podemos enmascarar interrupciones de la CPU hasta que esté listo para aceptar interrupciones.

(4) Pila de procesamiento de interrupción: A diferencia de las llamadas al sistema, las cuales solo pueden ocurrir en el Umode, las interrupcines pueden ocurrir en cualquiera, en el Umode o en el Kmode. El procesamiento de interrupción debe ser hecho en el Kmode. Cuando una interrupción de timer ocurre, la CPU guarda [flag, CS, PC] del punto interrumpido en la pila actual, la cual podría ser la pila de modo usuario o la pila de modo kernel. Entonces sigue el contenido del vector 8 para continuar la ejecución desde [\_tinth,0x1000] en el kernel MTX. En este momento, solamente CS=0x1000. Todos los otros registros de la CPU tienen aún los valores del punto interrumpido. Similar a \_int80h, podríamos hacer que \_tinth guarde todos los registros de la CPU en la pila actual. Entonces se debe tomar una decisión para establecer el ambiente de ejecución correcto.

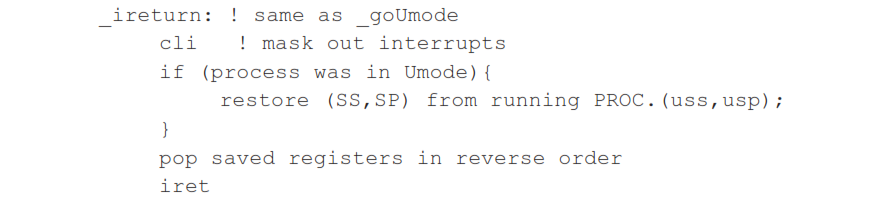
(4).1 Si la CPU estaba en el Umode antes de la interrupción, debemos cambiar DS, SS, ES al segmento de kernel 0x1000, guardar el par (SS,SP) del Umode interrumpido y switchear a la kstack del proceso que está en ejecución (running). Entonces llamar una función manejadora para continuar procesando la interrupción. En este caso, la situación es exactamente la misma que la de una syscall cuando el proceso entra a Kmode a través de INT 80. Así que podríamos guardar los registros de la CPU en el ustack del proc que está en ejecución (running), guarda (SS,SP) en su PROC.(uss,usp), y usar el kstack vacía de proc, i.e. hacer que sp apunte al extremo alto del kstack del proc que está corriendo (running).

(4).2 Si la CPU estaba en el Kmode antes de la interrupción, entonces no hay necesidad de cambiar DS, SS, ES, dado que ellos ya apuntan al segmento de kernel 0x1000. En este caso, debemos continuar usando la kstack del proc de running dado que esta no solamente contiene el contexto guardado del punto interrumpido sino también la historia de ejecución del proceso interrumpido en Kmode. Sería un desastre si tratáramos de usar el kstack de proc desde cero otra vez (proc’s kstack from scratch again).

¿Cómo sabemos en cuál modo estaba ejecutando la CPU antes de una interrupción? En algunas CPUs, un registro de status de la CPU tiene un campo de modo actual/previo, lo cual permite a un manejador de interrupción determinar en cuál modo ocurrió la interrupción. En otras CPUs, VyG, la x86 en modo protegido, la CPU switchea a la pila de modo kernel automáticamente después de una interrupción, así que no hay necesidad de decidir cuál pila usar. Dado que la CPU x86 en modo real no tiene esta capacidad tenemos que usar algunos otros medios para determinar si una interrupción ocurrió en Umode o en Kmode. En cualquier caso debemos asegurar que el procesamiento de la interrupción sea realizado en Kmode, tal que CS=segmento de código Kernel, DS=SS=segmento de data+stack del Kernel y la pila es el kstack del proc running.

(5). Después de establecer una pila de modo kernel apropiada, podríamos llamar a thandler() en C, la cual implementa el procesamiento de la interrupción de timer real. En thandler(), podríamos actualizar el tiempo actual, desplegar un reloj de pared, actualizar el tiempo de uso de CPU del proc running, ajustar prioridades de proceso y comenzar trabajo que depende del tiempo, etc. Entonces efectuar EOI para señalar el fin del procesamiento de interrupción.

(6). Cuando thandler() termina, regresa al lugar de donde la llamaron en \_tinth, lo cual regresa al punto interrumpido por



(7). Cuando el control entra en un manejador de interrupción, las interrupciones de la CPU

están enmascaradas. Después de guardar el contexto interrumpido, el manejador de interrupción típicamente desenmascara interrupciones para permitir que la CPU acepte otras interrupciones de mayor prioridad. Si es así, la CPU podría manejar otra interrupción antes de que el manejador de interrupción actual termine. Entonces, el procesamiento de interrupción podría ser anidado. Para interrupciones anidadas, la primera interrupción podría ocurrir en Umode. Cualesquiera interrupciones subsecuentes deben ocurrir en Kmode, i.e. en medio del procesamiento de una interrupción. En el último caso, continuamos con el uso del kstack de proc para guardar y regresar ahí al último punto interrumpido. Por lo tanto, nuestro esquema puede manejar interrupciones anidadas. La máxima profundidad de interrupciones anidadas está limitada por el número de IRQs diferentes, el cual es 15 en el caso de las PCs basadas en Intel x86 (Intel 64, 2011). Esto también implica que cada kstack de proc debe ser lo suficientemente grande como para contener hasta 15 capas de contextos interrumpidos guardados y sus marcos de pila. En lugar de requerir que cada proceso tenga una pila de kernel así de grande, algunas CPUs usan una única pila de interrupción hardware para procesamiento de interrupciones. En ese caso, cada kstack de proc solamente necesita espacio para mantener su contexto de ejecución en Kmode.

8.5 Implementación de Timer en MTX

En el kernel MTX, la implementación del procesamiento de interrupciones de timer es como sigue.

1 Enmascarar todas las interrupciones a través de lock(), lo cual ejecuta CLI.

2 Inicializar el kernel MTX como antes. Crear P0 y permitir a P0 bifurcar P1 usando kfork con una imagen de Umode. En kfork(), establecer el registro flag guardado a 0x0200 (I\_bit=1) tal que P1 correrá con las interrupciones habilitadas cuando éste empiece su ejecución.

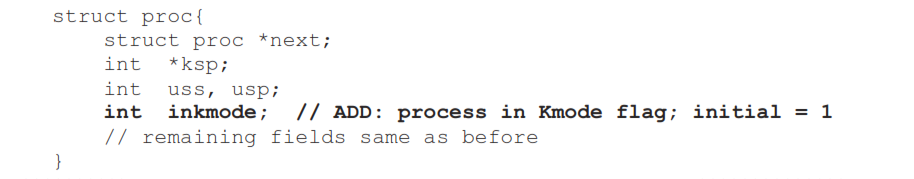
3 Establecer los vectores de interrupción de syscall y de interrupción de timer usando set\_vector(80,int80h); set\_vector(8,tinth);

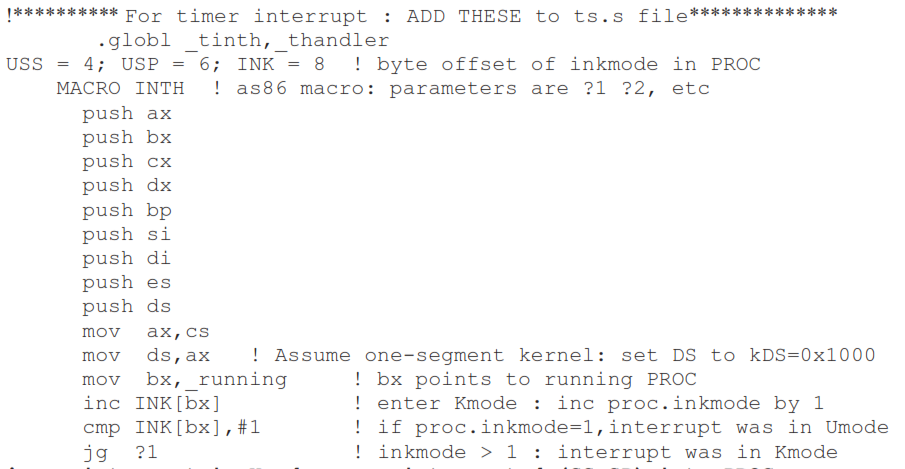
4 Inicializar y habilitar el timer con timer\_init(). Las interrupciones de timer ocurrirán inmediatamente pero están enmascaradas por ahora.

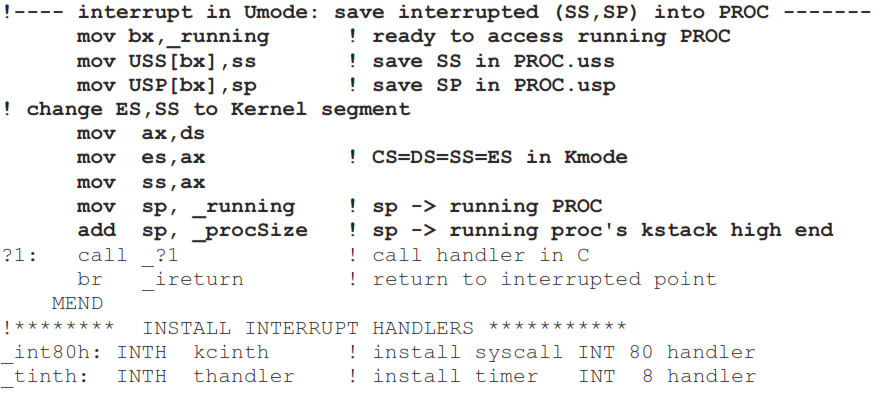
5 Hacer tswitch() a P1, el cual carga el registro flag de la CPU con I\_bit=1, permitiendo a la CPU aceptar interrupciones.

Los siguientes segmentos de código muestran la implementación de timer en MTX8.timer1, el cual está basado en MTX5.1. Los cambios principales en MTX8.timer1 están en el código ensamblador. Primero, con interrupciones, el código de tswitch() completo debe ser ejecutado con las interrupciones enmascaradas. Segundo, se usa una macro, INTH, para generar código de entrada y de salida apropiado para ambas, syscall e interrupciones de timer. Dado que la misma macro INTH también será utilizada más tarde para otras interrupciones, mostraremos el código de la macro INTH completo y lo explicaremos con más detalle.

Primero, agregamos un campo inkmode a cada estructura PROC en el offset de byte 8. Cuando un proceso es creado en el kernel, inicializamos su inkmode a 1. Cuando un proceso va al Umode, decrementamos su inkmode en 1. Cuando un proceso entra o reentra al kmode, incrementamos su inkmode en 1, etc. Entonces, podríamos usar el valor de inkmode del proc de running para determinar en cual modo ocurrió una interrupción. Si inkmode=1, ocurrió en Umode. Si inkmode>1, ocurrió en kmode. Alternativamente, podríamos también usar el registro de segmento guardado para determinar el modo de la CPU. Por ejemplo, si el CS guardado no es el segmento del kernel (0x1000), la interrupción debe haber ocurrido en el Umode. Lo siguiente muestra la estructura PROC modificada y la macro INTH.



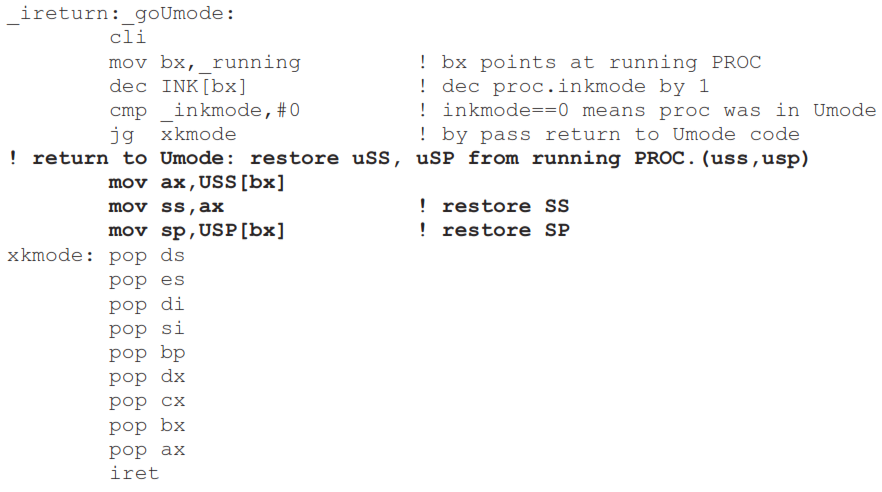


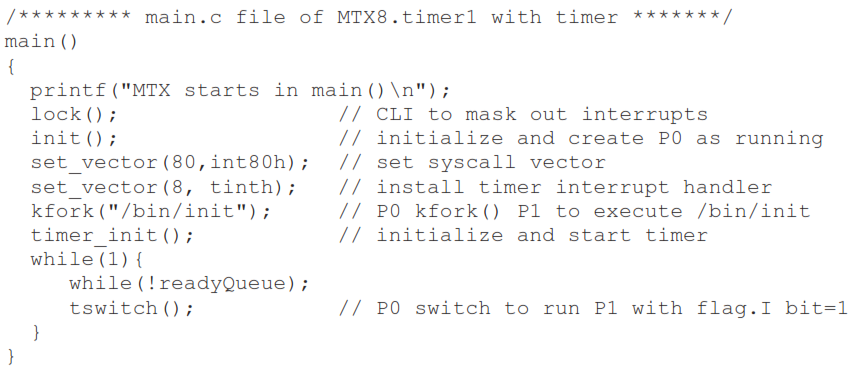


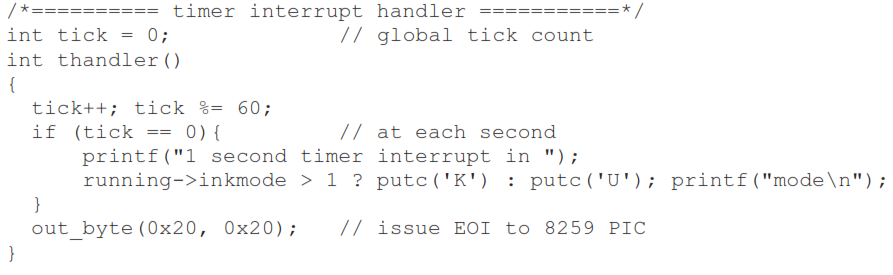
Los manejadores de interrupción son instalados con llamadas a la macro INTH de la forma



Cada llamada a la macro INTH genera un bloque de código ensamblador, en el cual el símbolo ?1 es remplazado con el parámetro Chandler. En el código ensamblador generado, este guarda todos los registros de la CPU en la pila actual. Entonces switchea DS al segmento de datos del kernel para acceder a la estructura PROC del proceso interrumpido. Usa el inkmode del proc para determinar si la interrupción ocurrió en Umode o en Kmode. Si la interrupción ocurrió en el Umode, las acciones son las mismas que las de INT 80. En este caso, usa la pila kstack vacía del proc de running para llamar al chandler. Si la interrupción ocurrió en el Kmode, continúa usando la pila kstack del proc de running para llamar a chandler directamente. Cuando la función chandler termina, regresa a \_ireturn, lo cual es lo mismo o algo análogo a lo que hace \_goUmode. Dependiendo del valor inkmode del proc, usa el marco de pila interrumpido guardado en ya sea el ustack o el kstack del proc para regresar al punto interrumpido.







MTX8.timer1 es un sistema muestra el cual demuestra un manejador de interrupción de timer simple. En cada segundo, despliega una línea para mostrar si la interrupción de timer ocurrió en el Kmode o en Umode. A pesar de las frecuentes interrupciones de timer el lector podría introducir comandos para correr el sistema como es usual. El lector podría también modificar el manejador de interrupción de timer para realizar otra tarea dependiente del tiempo, tal como

1 Desplegar un “reloj de pared” para mostrar el tiempo del sistema en el formato hr: min: sec.

2 Permitir a un proceso en ejecución dormir por unos cuantos segundos y despertar al proceso cuando su tiempo de dormir expira, etc.

8.6 Switcheo de Proceso por Rebanada de Tiempo

. . .

Page 239, de K.C. Wang, Designa and Implementation of the MTX Operating System, Springer International Publishing, 2015.