INSTITUTO POLITECNICO NACIONAL

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

Sistemas Operativos en Tiempo Real

Manejo de interrupciones en Marte OS

Presenta:

José Arturo Clemente Pérez

Maestro:

Lamberto Maza Casas

Grupo: 3MV10

Fecha: 12/12/2018

Contenido

Tabla de ilustraciones

Introducción

El concepto fundamental a manejar es el de interrupción. Una interrupción se describe como un mecanismo mediante el cual es posible interrumpir la ejecución del programa ejecutado por la CPU. La mayoría de los dispositivos utilizan interrupciones para notificar a la CPU que se ha producido un evento como:

- Existencia de un nuevo dato disponible
- Posibilidad de enviar un dato nuevo
- Cambio de una línea de estado
- Algún error
- Y más…

Tras una interrupción el programa permanece suspendido mientras se ejecuta la "rutina de servicio de interrupción" (ISR: Interrupt Service Routine) como se puede observar en la siguiente ilustración:

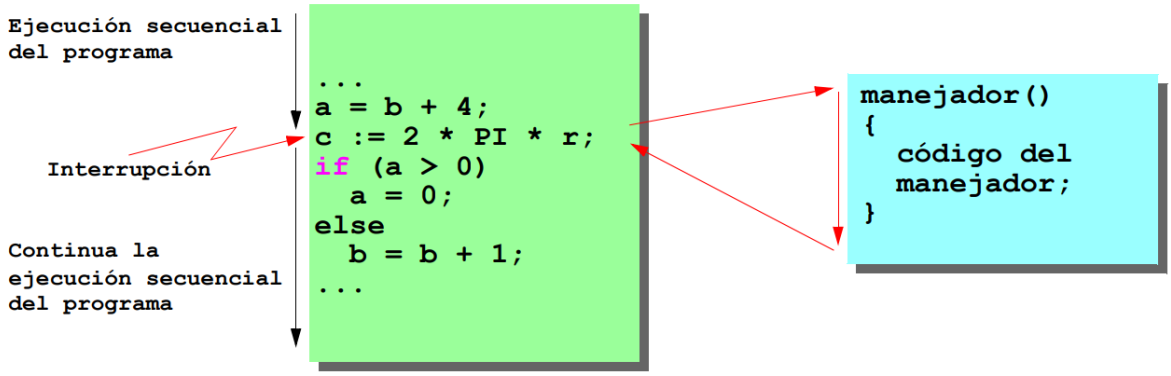

Ilustración 1 Rutina de servicio de interrupción [1]

Las interrupciones pueden ser de origen interno o externo a la CPU:

- excepción: interrupción generada como consecuencia del resultado de una ejecución de una instrucción.
- interrupción software: - generadas expresamente por el programa con una instrucción ensamblador.
- interrupción hardware: generada por un dispositivo.

Es necesario entender que las distintas interrupciones que se pueden producir en un computador se identifican mediante un número (tipo, prioridad de la interrupción), por ello se cuenta con la tabla de vectores de interrupción, la cual establece el enlace entre cada tipo de interrupción y su rutina de servicio de interrupción asociada (ISR), la siguiente imagen ilustra el control de rutinas desde el vector de interrupción:

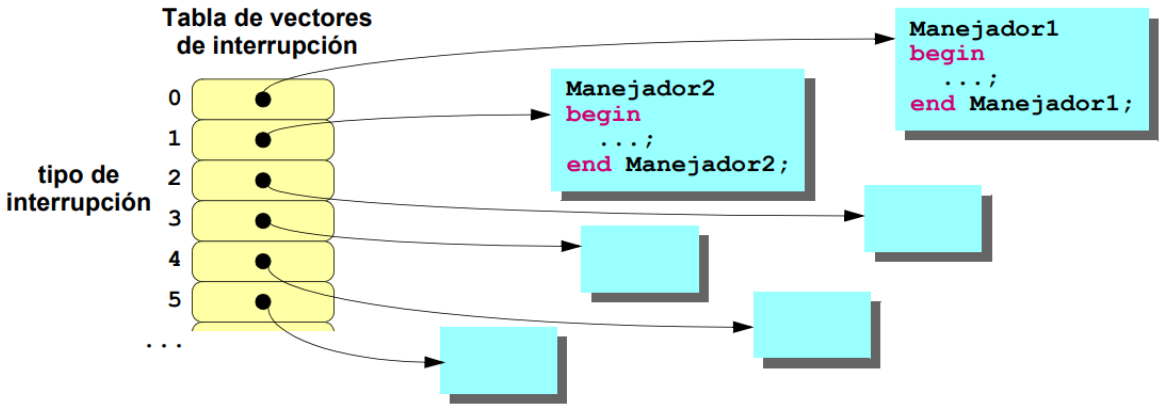

Ilustración 2 ISR asociada a cada tipo de interrupción [1]

Interrupciones gestionadas por Marte OS

MaRTE OS proporciona una interfaz para la gestión de interrupciones a nivel de aplicación en la que se definen operaciones para la habilitación y des-habilitación de interrupciones, así como para la instalación de manejadores de interrupción tradicionales que son ejecutados a la más alta prioridad del sistema inmediatamente después de producirse la interrupción. Existen dos versiones equivalentes de la interfaz, una en lenguaje Ada y otra en C. La interfaz Ada se encuentra definida en el paquete Hardware_Interrupts, mientras que la versión C se define en el fichero de cabeceras <hwinterrupts.h>. Las operaciones proporcionadas por ambas versiones de la interfaz son equivalentes una a una. [2]

Las interrupciones manejadas por el sistema Marte OS para una PC se muestran en la siguiente ilustración.

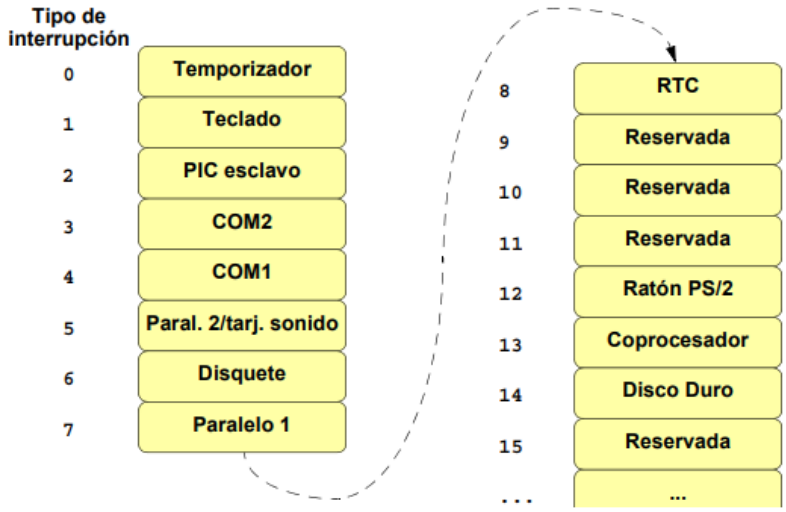

Ilustración 3 Tabla de vectores de interrupción para una PC [1]

Ahora bien, Marte OS permite dos modelos de gestión de interrupciones que son:

Thread asociada con ISR - utilizado cuando el driver tiene un thread dedicado (es el único thread que accede directamente al dispositivo), tiene la ventaja de ser más sencillo y eficiente, con la desventaja que sólo puede haber un thread asociado con cada interrupción

Sincronización por semáforos - utilizado cuando varios threads de usuario acceden directamente al dispositivo, tiene la ventaja que un thread puede esperar a la vez a varias interrupciones o bien, varios threads pueden esperar a la misma interrupción, los threads se encolan en el semáforo y son atendidos por prioridad.

Para la investigación realizada en este documento, se consideró únicamente el modelo de una Thread asociada con ISR, a continuación, se muestra una ilustración de este modelo.

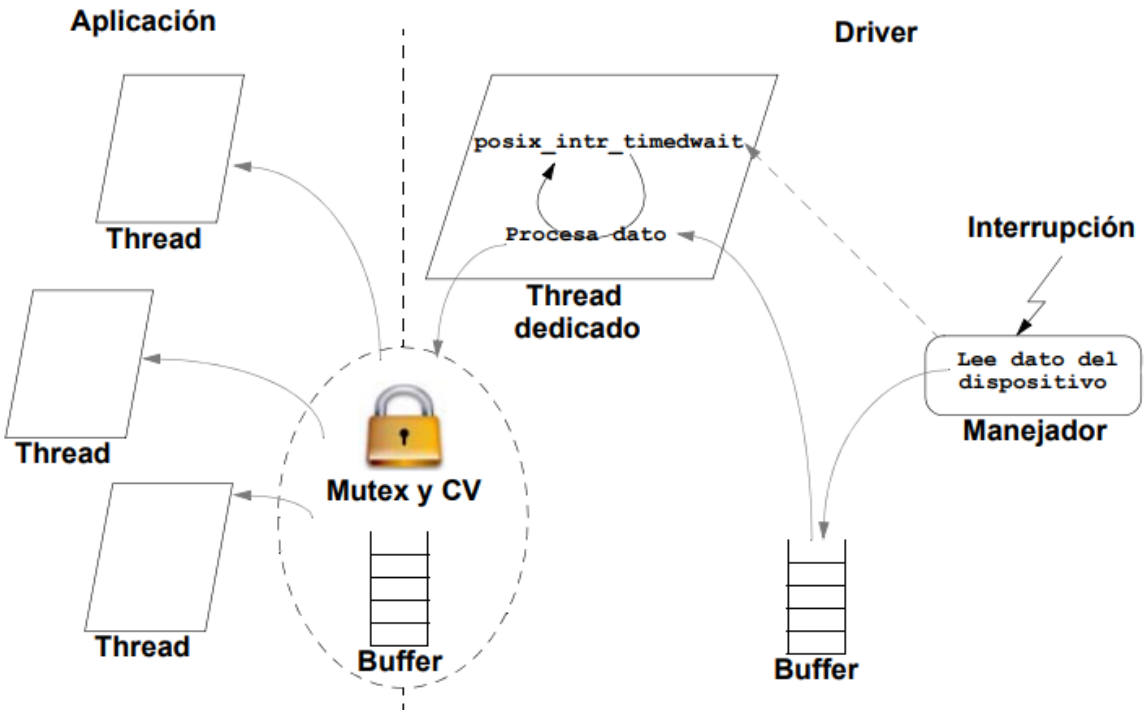

Ilustración 4 Thread asociada con ISR

Interfaz para la gestión de interrupciones

MaRTE OS proporciona una interfaz no estándar (en) que permite:

• I**nstalar y desinstalar manejadores de interrupción** - posix_intr_associate() y posix_intr_disassociate()

• **Bloquear y desbloquear interrupciones** - posix_intr_lock() y posix_intr_unlock()

• **Esperar interrupciones** (Thread asociada con ISR) - posix_intr_timedwait()

La identificación de las interrupciones se puede observar en la siguiente ilustración:

TIMER_HWINTERRUPT	temporizador
KEYBOARD_HWINTERRUPT	teclado
SERIAL1_HWINTERRUPT	Puerto serie 1
PARALLEL1_HWINTERRUPT	Puerto paralelo 1
DISKETTE_HWINTERRUPT	Disquete
COPROCESSOR_HWINTERRUPT	Coprocesador matemático
...	Resto de interrupciones

Ilustración 5 Fuentes de interrupción [1]

Utilizando <intr.h>, se define el tipo intr_t para identificar las fuentes de interrupciones existentes en el sistema.

La interfaz anterior se refiere al lenguaje C, sin embargo, se puede traducir para el lenguaje ADA como se implementó en el salón de clases, la intención es utilizar las interrupciones de un dispositivo externo para realizar una acción en la PC, la siguiente sección se muestra el ejemplo dado en clase y además se da un código para interpretar las interrupciones por el teclado manejado por el sistema Marte OS.

Código propuesto y conclusiones

Ejemplo utilizado en clase, encendido de leds por manejo de interrupciones del puerto serie.

```
1. ------------------------------------------------------
2. --  This Example was used with a set of LEDs attached to the
   parallel port
3. --  in order to show a first example of an interaction with the
   hardware
4. --
5. --  Build command: mgnatmake leds_parallel_port.adb
6. --
7. --  daniel.sangorrin@gmail.com
8. --
9. --  mgnatmake leds_parallel_port.adb -Imarte_src_dirs
10.      ------------------------------------------------------
11.      --  for outb, io_port
12.      with MaRTE.HAL.IO; use MaRTE.HAL.IO;
13.      --  for IRQs
14.      with POSIX_Hardware_Interrupts;
15.      with System;
```

```
16.      with MaRTE.Direct_IO;
17.
18.      procedure LEDs_Parallel_Port is
19.
20.         package PHI renames POSIX_Hardware_Interrupts;
21.         ------------------
22.         -- 1) Constants --
23.         ------------------
24.         PP_IRQ         : constant PHI.Hardware_Interrupt :=
   PHI.PARALLEL1_INTERRUPT;
25.         PP_BASE_REG    : constant IO_Port := 16#378#; --
26.         PP_DATA_REG    : constant IO_Port := PP_BASE_REG + 0; --
   Data port offset
27.         PP_STATUS_REG  : constant IO_Port := PP_BASE_REG + 1; --
   Status port offset
28.         PP_CONTROL_REG : constant IO_Port := PP_BASE_REG + 2; --
   Control port offset
29.         PP_IRQ_ENABLE  : constant := 2#0001_0000#;
30.
31.         --------------------------
32.         -- PP_Interrupt_Handler --
33.         --------------------------
34.         Notify_IRQ : Boolean := True;
35.         function PP_IRQ_Handler
36.           (Area : in System.Address;
37.            Intr : in PHI.Hardware_Interrupt)
38.           return PHI.Handler_Return_Code is
39.         begin
40.            MaRTE.Direct_IO.Put ("irq");
41.            if Notify_IRQ = True then
42.               Notify_IRQ := False;
43.               return PHI.POSIX_INTR_HANDLED_NOTIFY;
44.            else
45.               return PHI.POSIX_INTR_HANDLED_DO_NOT_NOTIFY;
46.            end if;
47.         end PP_IRQ_Handler;
48.         Intr    : PHI.Hardware_Interrupt;
49.         Handler : PHI.Interrupt_Handler;
50.
51.      begin
52.         PHI.Associate (PP_IRQ, PP_IRQ_Handler'Unrestricted_Access,
   System.Null_Address, 0);
53.         --  Enable interrups in the printer port
54.         Outb_P (PP_CONTROL_REG, PP_IRQ_ENABLE);
55.         --  Enable the interrupt
56.         PHI.Unlock (PP_IRQ);
57.         loop
58.            Notify_IRQ := True;
59.      --       MaRTE.Direct_IO.Put ("ANTES de PHI.Wait (Intr,
   Handler)");
60.      --       PHI.Wait (Intr, Handler);
61.      --       MaRTE.Direct_IO.Put ("Despues de PHI.Wait (Intr,
   Handler)");
62.            MaRTE.Direct_IO.Put ("Se envian datos al puerto
   paralelo");
63.            Outb_P (PP_DATA_REG, 2#00_001_001#); delay (0.5);
64.            Outb_P (PP_DATA_REG, 2#00_010_010#); delay (0.5);
```

```
65.            Outb_P (PP_DATA_REG, 2#00_100_100#); delay (0.5);
66.            Outb_P (PP_DATA_REG, 2#00_010_010#); delay (0.5);
67.            Outb_P (PP_DATA_REG, 2#00_001_001#); delay (0.5);
68.            Outb_P (PP_DATA_REG, 2#00_000_000#);
69.         end loop;
70.      end LEDs_Parallel_Port;
```

Manejo de interrupciones del teclado

```
1. --  for outb, io_port
2. with MaRTE.HAL.IO; use MaRTE.HAL.IO;
3. --  for IRQs
4. with POSIX_Hardware_Interrupts;
5. with System;
6. with MaRTE.Direct_IO;
7.
8. procedure Observado_Teclado is
9.
10.         package PHI renames POSIX_Hardware_Interrupts;
11.         ------------------
12.         -- 1) Constants --
13.         ------------------
14.         PP_IRQ         : constant PHI.Hardware_Interrupt :=
    PHI.KEYBOARD_INTERRUPT;
15.      --    PP_BASE_REG    : constant IO_Port := 16#378#; --
16.      --    PP_DATA_REG    : constant IO_Port := PP_BASE_REG + 0; --
    Data port offset
17.      --    PP_STATUS_REG  : constant IO_Port := PP_BASE_REG + 1; --
    Status port offset
18.      --    PP_CONTROL_REG : constant IO_Port := PP_BASE_REG + 2; --
    Control port offset
19.      --    PP_IRQ_ENABLE  : constant := 2#0001_0000#;
20.
21.         -------------------------
22.         -- PP_Interrupt_Handler --
23.         -------------------------
24.         Notify_IRQ : Boolean := True;
25.         function PP_IRQ_Handler
26.           (Area : in System.Address;
27.            Intr : in PHI.Hardware_Interrupt)
28.           return PHI.Handler_Return_Code is
29.         begin
30.            MaRTE.Direct_IO.Put ("irq");
31.            if Notify_IRQ = True then
32.               Notify_IRQ := False;
33.               return PHI.POSIX_INTR_HANDLED_NOTIFY;
34.            else
35.               return PHI.POSIX_INTR_HANDLED_DO_NOT_NOTIFY;
36.            end if;
37.         end PP_IRQ_Handler;
38.         Intr    : PHI.Hardware_Interrupt;
39.         Handler : PHI.Interrupt_Handler;
40.
41.      begin
42.         PHI.Associate (PP_IRQ, PP_IRQ_Handler'Unrestricted_Access,
    System.Null_Address, 0);
```

```
43.
44.          --  Enable the interrupt
45.          PHI.Unlock (PP_IRQ);
46.          loop
47.             Notify_IRQ := True;
48.       --       MaRTE.Direct_IO.Put ("ANTES de PHI.Wait (Intr,
   Handler)");
49.       --       PHI.Wait (Intr, Handler);
50.       --       MaRTE.Direct_IO.Put ("Despues de PHI.Wait (Intr,
   Handler)");
51.             MaRTE.Direct_IO.Put ("Se detecto una tecla pulsada en
   el teclado");
52.          end loop;
53.       end Observando_Teclado;
```

Esta es la conclusión del trabajo, con la implementación anterior se puede manejar una interrupción a la vez de cualquier medio disponible por el gestor de interrupciones de marte OS, los cuales se mostraron en las ilustraciones 3 y 4, a cada evento externo ocurrido, se puede realizar una acción en la computadora desde leer los datos recibidos hasta modificar otras áreas del código, en el código anterior, se modifica una parte del programa sin embargo aún no se logra leer los datos recibidos dando pauta para una investigación futura.

Referencias

[1] M. Aldea, M. González. (2014). Gestión de Interrupciones en MaRTE OS. 10 de diciembre de 2018, de Universidad de Cantabria, Sitio web: https://www.istr.unican.es/asignaturas/PTR/PlatTR_02_interrupt_3en1.pdf

[2] Mario Aldea Rivas. (2002). Planificación de Tareas en Sistemas Operativos de Tiempo Real Estricto para Aplicaciones Empotradas. Universidad de Cantabria: Santander.