Master en Computación

Plataformas de Tiempo Real

POSIX Avanzado y Extensiones

Tema 2. Gestión de Interrupciones en MaRTE OS

2.1 Interrupciones

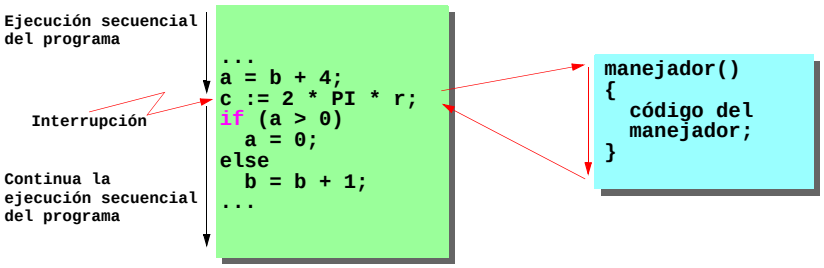
***Interrupción*: mecanismo mediante el cual es posible interrumpir la ejecución del programa ejecutado por la CPU**

La mayoría de los dispositivos utilizan *interrupciones* para notificar a la CPU que se ha producido un evento:

- **nuevo dato disponible**
- **posibilidad de enviar nuevo dato**
- **cambio en una línea de estado**
- **error**
- **etc.**

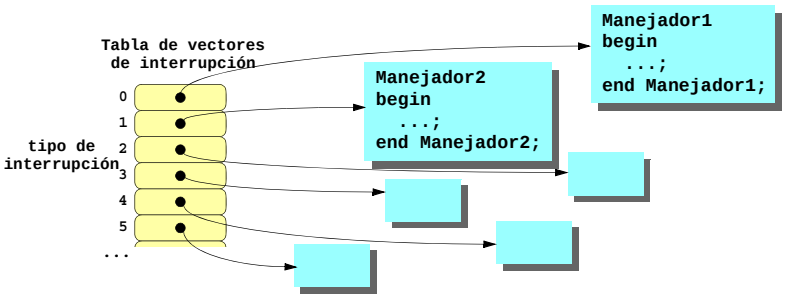
Conceptos fundamentales

Tras la interrupción el programa permanece suspendido mientras se ejecuta la *rutina de servicio de interrupción* (*ISR*) (también denominada *manejador de la interrupción*)



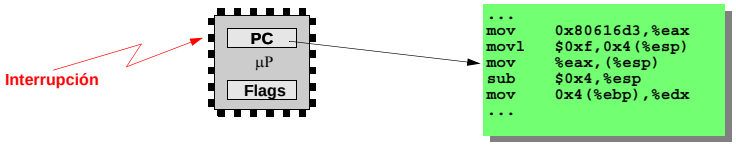
Las distintas interrupciones que se pueden producir en un computador se identifican mediante un número (*tipo de la interrupción*)

La *tabla de vectores de interrupción* establece el enlace entre cada tipo de interrupción y su ISR asociada

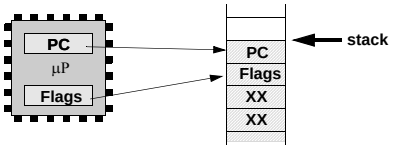


Ciclo de atención a interrupción

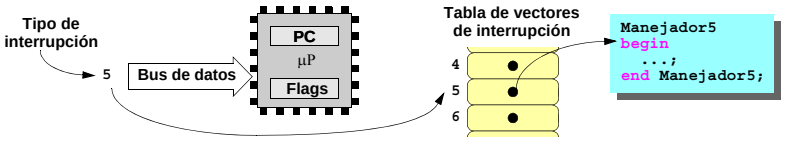
1. **Se genera una interrupción, el procesador termina la instrucción ensamblador que estaba ejecutando**



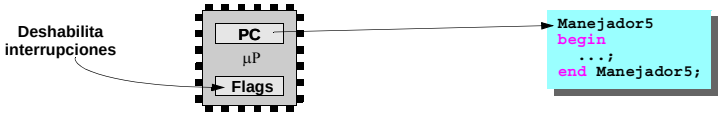
2. **Se salva en el stack el estado del procesador (contador de programa y registro de estado)**



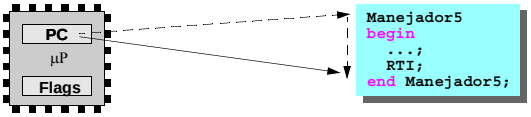
3. La CPU lee el tipo de interrupción y obtiene la dirección de su ISR utilizando la tabla de vectores de interrupción



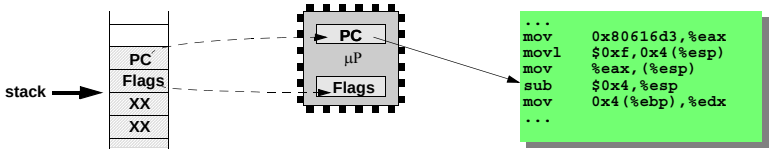
4. Se carga la dirección de comienzo de la ISR en el PC y se deshabilitan nuevas interrupciones



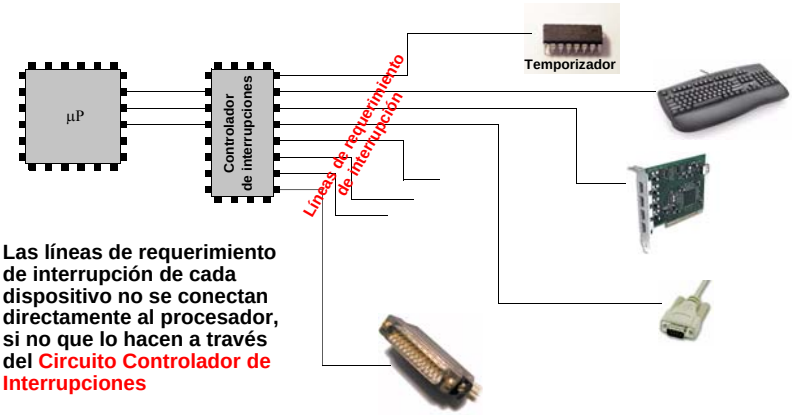
5. Se ejecuta la ISR hasta llegar a la instrucción de retorno de interrupción (RTI)



6. La instrucción RTI recupera del stack el estado original de la CPU, con lo que el procesador continua la ejecución del programa interrumpido

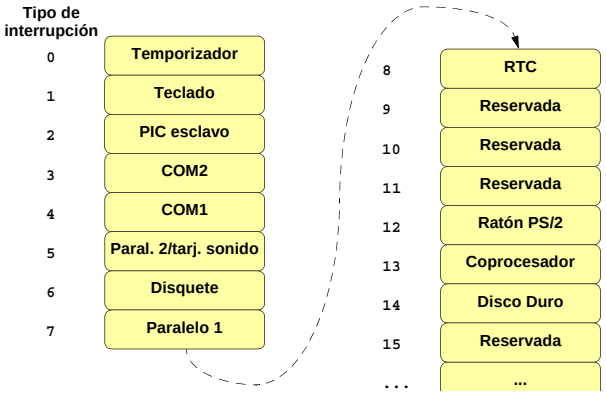


Controlador de interrupciones

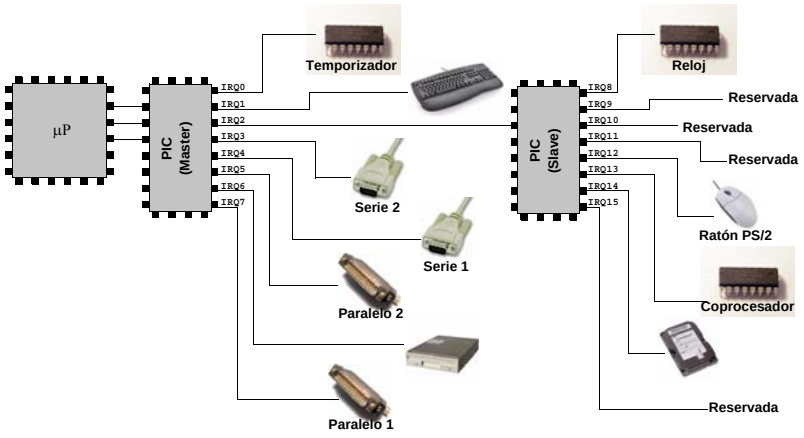


Las líneas de requerimiento de interrupción de cada dispositivo no se conectan directamente al procesador, si no que lo hacen a través del Circuito Controlador de Interrupciones

Tabla de vectores de interrupción de un PC



Configuración de las interrupciones hardware en un PC estándar



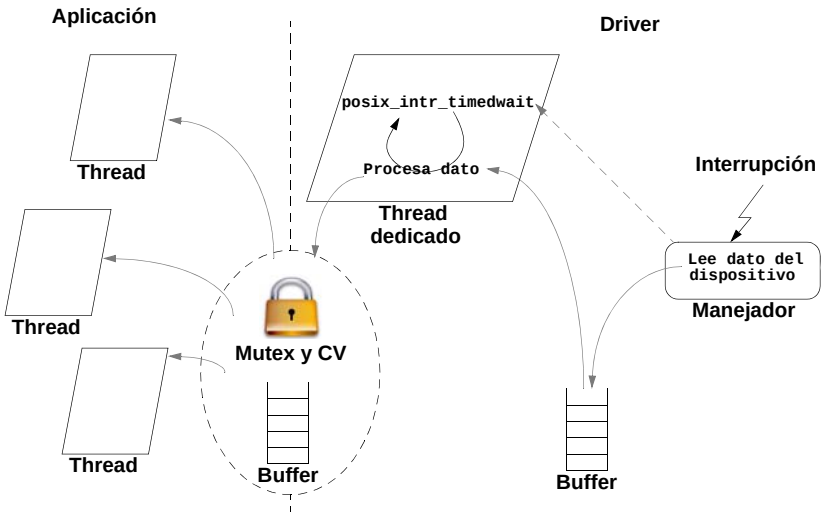
2.2 Modelos de gestión de interrupciones en MaRTE OS

MaRTE OS permite dos modelos de gestión de interrupciones:

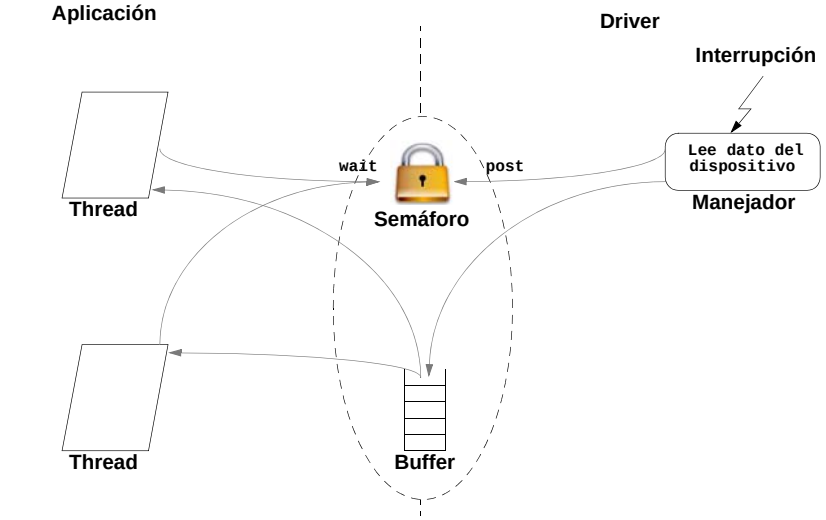
- **Thread asociada con ISR**
  - **utilizado cuando el driver tiene un thread dedicado (es el único thread que accede directamente al dispositivo)**
  - ✓ **más sencillo y eficiente**
  - ✗ **sólo puede haber un thread asociado con cada interrupción**

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

Thread asociada con ISR



Sincronización por semáforo



2.3 Interfaz para la gestión de interrupciones

La gestión de interrupciones no está estandarizada en POSIX

MaRTE OS proporciona una interfaz no estándar (en `<intr.h>`) que permite:

- **instalar 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()`**

(los nombres comienzan por "`posix_`" porque la interfaz fue propuesta para una futura ampliación del estándar POSIX)

Fuentes de interrupción

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

Además se proporcionan valores constantes de este tipo para las distintas fuentes. Por ejemplo, en un PC:

`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

Manejadores de interrupción

Los manejadores de interrupción son funciones con el siguiente prototipo

```
int intr_handler (void * area, intr_t intr)
```

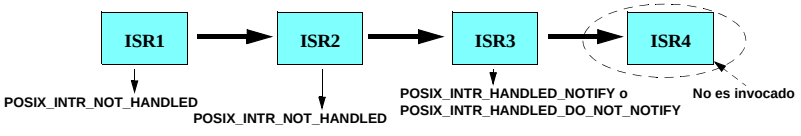
- **`area` permite identificar una región de memoria mediante la cual el manejador y la aplicación pueden compartir datos**
- **`intr` identifica la fuente de interrupción que ha provocado la invocación del manejador (útil cuando se utiliza el mismo manejador para varias interrupciones)**

Un manejador puede sincronizarse con los threads de la aplicación utilizando semáforos definidos en `area`

- **el manejador podrá invocar `sem_post()` sobre esos semáforos**

Un manejador debe retornar uno de los valores siguientes:

- **`POSIX_INTR_HANDLED_NOTIFY`: el manejador ha atendido la interrupción y, si hay algún thread esperando, deberá ser activado por el sistema operativo**
- **`POSIX_INTR_HANDLED_DO_NOT_NOTIFY`: el manejador ha atendido la interrupción pero el thread NO debe activarse**
- **`POSIX_INTR_NOT_HANDLED`: el manejador NO ha atendido la interrupción; si hay otros manejadores asociados el sistema operativo deberá invocar al siguiente**



Asociación de Manejadores

Asociar un manejador con una fuente de interrupción y un thread:

```
#include <intr.h>
int posix_intr_associate
   (intr_t intr,
    int (*intr_handler)(void * area, intr_t intr),
    void * area,
    size_t areasize);
```

- `intr`: fuente de interrupciones
- `intr_handler`: manejador
- `area`: región de memoria de `areasize` bytes para compartir datos entre el manejador y la aplicación

El manejador se asocia con el thread que invoca esta función

Asociación de Manejadores (cont.)

Romper una asociación:

```
#include <intr.h>
int posix_intr_disassociate
   (intr_t intr,
    int (*intr_handler)(void *area, intr_t intr));
```


Bloqueo y desbloqueo de interrupciones

Bloqueo y desbloqueo de interrupciones:

```
#include <intr.h>
int posix_intr_lock (intr_t intr);
int posix_intr_unlock (intr_t intr);
```

La llamada a `posix_intr_lock()` permite bloquear la interrupción identificada por `intr`

- **los manejadores asociados no se ejecutan mientras la interrupción esté bloqueada**
- **Su ejecución permanece pendiente hasta que se invoca `posix_intr_unlock()`**

No se especifica si sucesivas interrupciones producidas con la interrupción bloqueada se encolan o se pierden

Espera de interrupciones

Un thread asociado con una o varias interrupciones (utilizando `posix_intr_associate()`) puede esperar la ejecución de cualquiera de sus manejadores mediante la función:

```
#include <intr.h>
int posix_intr_timedwait (int flags,
   const struct timespec *abs_timeout,
   intr_t *intr,
   int (**intr_handler) (void *area, intr_t intr))
```

- **`flags`: opciones dependientes de la implementación**
- **`abs_timeout`: tiempo límite absoluto (basado en `CLOCK_REALTIME`), si es `NULL` espera para siempre**
- **`intr`: fuente de interrupción que ha causado la activación**
- **`intr_handler`: manejador que ha causado la activación**

Uso habitual en drivers: Thread asociada con ISR

- **open (o `create`)**
  - **Crea el thread dedicado, el cual:**
    - **Asocia el manejador (`posix_intr_associate`)**
    - **Espera la ejecución del manejador(`posix_intr_timedwait`)**
    - **Bloquea y desbloquea interrupciones para sincronizarse con el manejador (`posix_intr_lock`, `posix_intr_unlock`)**
- **`read`/`write`**
  - **No gestionan las interrupciones**
- **close**
  - **Bloqueo final de la interrupción (`posix_intr_lock`)**
  - **Finaliza el thread dedicado**

Uso habitual en drivers: Sincronización por semáforo

- **open (o `create`)**
  - **Asocia el manejador (`posix_intr_timedwait`)**
  - **Desbloqueo inicial de la interrupción (`posix_intr_unlock`)**
- **`read`/`write`**
  - **Espera en el semáforo (que es señalizado por el manejador)**
  - **Bloquea y desbloquea interrupciones para acceder a los datos compartidos con el manejador (`posix_intr_lock`, `posix_intr_unlock`)**
- **close**
  - **Bloqueo final de la interrupción (`posix_intr_lock`)**

2.4 Ejemplo: Espera de interrupción

```
#include <intr.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <misc/error_checks.h>
#include <misc/timespec_operations.h>


// Interrupt handler
int interrupt_handler(void * area, intr_t intr)
{
  printc("In interrupt handler\n");
  return POSIX_INTR_HANDLED_NOTIFY;
}
```



```
int main()
{
  const struct timespec rel_timeout = {5, 0};
  struct timespec timeout;
  intr_t intr;
  int (*handler)(void * area, intr_t intr);

  // Install interrupt handler
  CHK( posix_intr_associate(FUENTE_INTERRUPCION, interrupt_handler, NULL, 0) );

  //  Enable the interrupt in the computer
  CHK( posix_intr_unlock (FUENTE_INTERRUPCION) );

  //  Read inital time an wait interrupts in a loop
  CHKE( clock_gettime(CLOCK_REALTIME, &timeout) );
  while(1) {
    // timeout calculation
    add_timespec(&timeout, &timeout, &rel_timeout);

    // Wait for interrupt
    CHK( posix_intr_timedwait(0, &timeout, &intr, &handler) );
    printf("After interrupt\n");
  }

  return 0;
}
```


2.5 Ejemplo: Sincronización con semáforos

```
#include <intr.h>
#include <stdio.h>
#include <semaphore.h>
#include <string.h>
#include <stdlib.h>
#include <misc/error_checks.h>
#include <sys/pio.h>


// datos compartidos entre el manejador y el thread
typedef struct {
  sem_t sem;
  unsigned char dato;
} area_manejador_t;

area_manejador_t area_manejador;
```

```
// manejador de interrupción
int manejador_intr (void * area, intr_t intr) {
  printc("En el manejador\n");

  // lee dato del dispositivo
  ((area_manejador_t *)area)->dato = inb(0x378);

  // Señaliza el semáforo
  CHKE( sem_post(&((area_manejador_t *)area)->sem) );

  return POSIX_INTR_HANDLED_DO_NOT_NOTIFY;
}

// lee dato (función llamada por los threads)
// (la operación read de un driver se comportaría de forma similar)
unsigned char lee_dato() {
  // Si no hay datos, los threads se encolan en el semáforo
  CHKE( sem_wait(&area_manejador.sem) );

  // lee el dato
  // cuando el dato a leer no es atómico es necesario mantener la interrupción
  // deshabilitada (no en este caso, aún así se hace para que sirva de ejemplo)
  CHKE( posix_intr_lock(FUENTE_INTERRUPCION) );
  unsigned char d = area_manejador.dato;
  CHKE( posix_intr_unlock(FUENTE_INTERRUPCION) );

  return d;
}
```



```
int main ()
{
  unsigned char dato;

  // inicializa el semáforo
  CHKE( sem_init(&area_manejador.sem, 0, 0) );

  // instala el manejador de interrupción
  CHKE( posix_intr_associate(FUENTE_INTERRUPCION, manejador_intr,
                             &area_manejador, sizeof(area_manejador_t)) );

  //  habilita la interrupción
  CHKE( posix_intr_unlock(FUENTE_INTERRUPCION) );

  // lee datos del dispositivo
  while(1) {
    // espera un nuevo dato
    dato = lee_dato();
    printf("Leído %c\n", dato);
  }

  return 0;
}
```