ESTRUCTURA DE COMPUTADORES
Curso 2016/2017

Práctica 15: Llamadas al sistema operativo (II)

Esta práctica es una continuación de la práctica anterior, por tanto sus objetivos son los mismos. Se tomará como punto de partida el manejador Mimos_v3.handler, donde se tienen implementadas las cuatro llamadas al sistema *get_version*, *print_char*, *get_time* y *wait_time*. Y se pedirá la implementación de una nueva llamada al sistema, *read_char* y la modificación de *print_char*.

Material

- La versión del simulador PCSIM utilizada en la práctica anterior.
- La versión preliminar del manejador *MIMOSv3.handler*.
- Archivo de prueba: *Usuario.s*

Los procesos en MiMoS

Se recuerda que en esta versión del manejador se había dotado al sistema de capacidad mutliproceso, de modo que hay también definido un **proceso ocioso** como el siguiente:

```
proceso_ocioso: # proceso ocioso del sistema
                b proceso_ocioso
```

Este proceso siempre está activo y su contexto es mínimo, porque no utiliza ningún registro del procesador y, cuando se ejecuta, el contador de programa apunta constantemente a la dirección `proceso_ocioso.`

Como el código del manejador MiMoS se invoca siempre a través de una excepción, se debe determinar al final de su ejecución a qué instrucción se debe retornar.

Recuerde que al final del código manejador de excepciones (etiqueta retexc) está el código que realiza la **gestión de procesos** y deja la dirección de retorno en *$k0*. Sólo hay dos opciones: si el proceso usuario está `listo`, entonces entra en ejecución; en cualquier otro caso entra el proceso ocioso:

Si (`estado` = `listo`*)*
    $k0 = dirección de retorno del proceso usuario
si no
    $k0 = `proceso_ocioso`
fin si

Además se recuerda también que el **cambio de contexto** no es necesario, porque sólo hay que mantener el contexto del proceso principal.

Tarea 1. Manejador versión 3 (MiMoS v.3). Función read_char.

La versión 3 de MiMoS se presenta con las llamadas al sistema *get_version, get_time, print_char* y *wait_time.* El servicio *read_char* o lectura de caracteres mostrado en la **Tabla 1** es el que se tiene que implementar en esta práctica, y en el siguiente apartado se modificará el servicio *print_char*.

La llamada al sistema *read_char* tendrá que poner en estado de espera el proceso, hasta que el teclado esté disponible. Será la interrupción del teclado la que cambie el estado del proceso en espera y proporcione el valor en el registro $a0.

Función	Código	Argumentos	Resultados
`get_version`	*$v0* = 90		Número de versión (en *$v0*)
`get_time`	*$v0* = 91		Tiempo en segundos (en *$v0*)
`wait_time`	*$v0* = 92	$a0 = tiempo en segundos	
`print_char`	*$v0* = 11	$a0 = carácter a imprimir	
`read_char`	*$v0* = 12		$a0 = carácter leído

Tabla 1: Servicios a implementar en MiMoS v.3. Los servicios get_version, get_time, print_char y wait_time están ya implementados en el manejador. La llamada read_char deberá ser implementada.

El teclado debe tener la interrupción habilitada sólo desde el momento en que el programa efectúe la llamada al sistema ***read_char*** hasta que el periférico esté disponible. Para simplificar la gestión del teclado, convendrá tener su línea de interrupción *int0** desenmascarada permanentemente en el registro de estado del coprocesador y realizar las operaciones de habilitación/inhibición sobre la interfaz del teclado.

La secuencia de eventos que se suceden durante la espera de la función *read_char* se muestra en la figura 1. Como en el la práctica enterior, la idea consiste en cambiar el estado del proceso que llama a la función *read_char* desde el valor `LISTO` al valor `ESPERANDO_LEER`. Con esta acción se hará que el manejador no retorne al proceso principal, sino que retornará al proceso ocioso. La rutina de interrupción del teclado será la encargada de restaurar el estado al valor `LISTO`, lo que provocará el retorno al proceso principal.

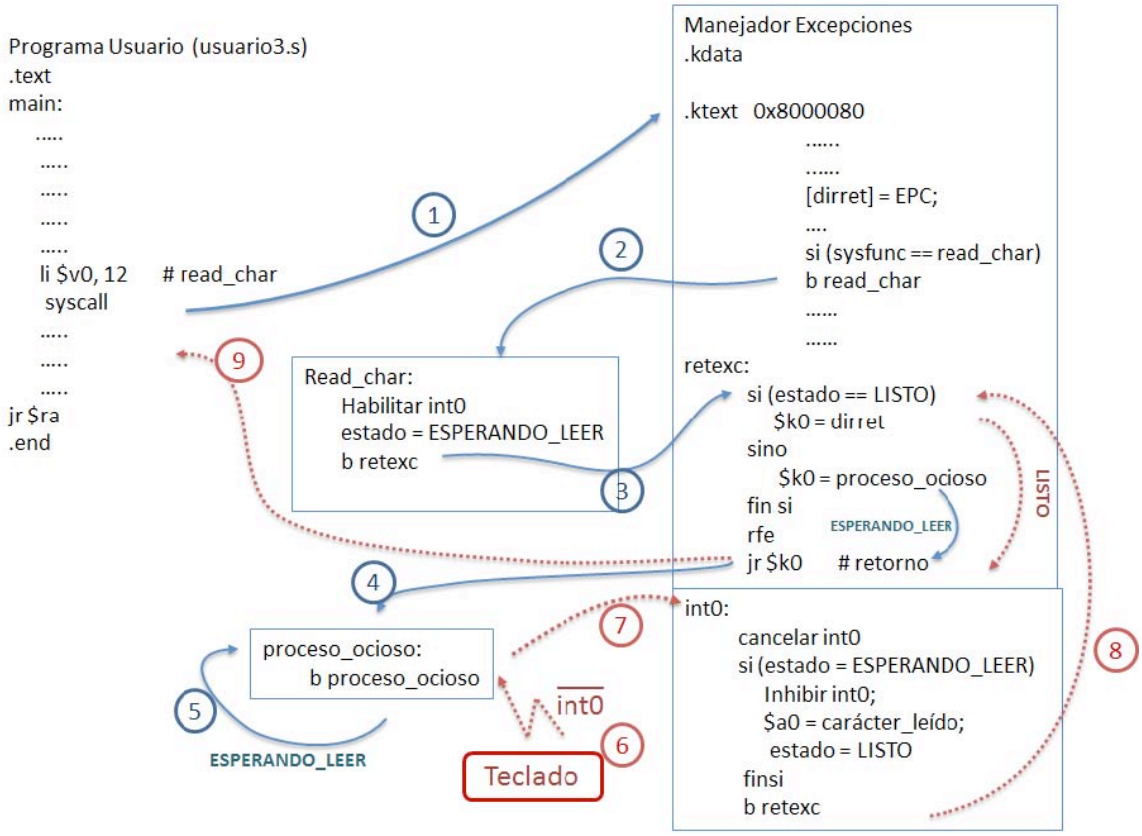

Figura 1 – Conmutación de procesos en la llamada a *read_char*.

Realice las siguientes modificaciones en *MiMoSv3.handler*:

► Implemente la llamada *read_char*. El tratamiento debe poner el proceso usuario en estado `ESPERANDO_LEER` y habilitar la interrupción en la interfaz de teclado. Para ello deberá definir la constante `ESPERANDO_LEER` al principio donde se encuentran las otras constantes de estado.

```
read_char:
        Habilitar interrupción de teclado;
        estado = ESPERANDO_LEER;
        b retexc
```

Cuestión 1. Escriba el código correspondiente a la llamada al sistema *read_char*.

read_char:

► Escriba el tratamiento de la interrupción de teclado (etiqueta `int0`). El tratamiento sólo se debe aplicar si el proceso usuario se encuentra en estado `ESPERANDO_LEER` y consistirá en dejar el carácter leído en *$a0*, inhibir la interrupción de teclado y poner el proceso usuario en estado `LISTO`.

Note que este servicio se limita a leer el carácter del teclado sin imprimirlo por la pantalla.

```
int0:
        si (estado = ESPERANDO_LEER)
                $a0 = carácter leído;
                cancelar e inhibir interrupción de teclado;
                estado = LISTO;
        fin si
        b retexc
```

Cuestión 2. Escriba el código correspondiente a la interrupción de teclado, desde la etiqueta *int0*.

```
int0:
```

► Programe la máscara de interrupciones en la sección de inicio del sistema, línea INT0* habilitada (además de la del reloj).
Escriba el valor de la máscara en hexadecimal.

► Pruebe el manejador utilizando el programa *usuario.s* con el siguiente esquema:

Saludo inicial;
*Llamar a **get_time**;*
Escribir en la consola el tiempo actual;
*Llamar **read_char**;*
Escribir en la consola el carácter leído;
*Llamar a **get_time**;*
Escribir en la consola el tiempo actual;

Tarea 2. Manejador versión 3 (MiMoS v.3). Función print_char.

Se ha de modificar el servicio de impresión de carácter *print_char* (**Tabla 2**) para que se sincronice por interrupción, ahora está implementado por consulta del estado. El mecanismo deber ser como el de la función *read_char*: la interrupción de consola sólo debe estar habilitada desde el momento en que el programa llama a la función ***print_char*** hasta el momento en que el periférico está listo, dejando el proceso en *ESPERANDO_ESCRIBIR*. Y la interrupción de la consola será la que cambie el estado del proceso y realice la impresión.

Función	Código	Argumentos	Resultados
`print_char`	*$v0* = 11	$a0 = carácter a imprimir	

Tabla 2: Nuevo servicio a implementar print_char.

Realice la siguiente lista de actividades:

► Implemente el código del servicio *print_char*. El nuevo tratamiento debe poner el proceso usuario en estado *ESPERANDO_ESCRIBIR* y habilitar la interrupción en la interfaz de consola.

```
print_char:
        habilitar interrupción de consola;
        estado = ESPERANDO_ESCRIBIR;
        b retexc
```

Cuestión 3

Escriba el código correspondiente a la llamada al sistema *print_char*.

```
print_char:
```

► Escriba el tratamiento de la interrupción de consola (etiqueta `int1`), que sólo se debe aplicar si el proceso se encuentra en estado `ESPERANDO_ESCRIBIR`. El código ha de copiar *$a0* en el registro de datos, inhibir las interrupciones de consola y poner el proceso usuario en estado `LISTO`.

```
int1:
        si (estado = ESPERANDO_ESCRIBIR)
                carácter a escribir = $a0;
                cancelar e inhibir interrupción de consola;
                estado = LISTO;
        fin si
        b retexc
```

Cuestión 4

Escriba el código correspondiente a la interrupción de teclado, desde la etiqueta int1.

```
int1:
```

► Programe la máscara de interrupciones adecuada en la sección de inicio del sistema. Para probar el nuevo tratamiento se puede utilizar cualquier programa de usuario anterior, pues todos ellos utilizan la función `print_char`

Escriba el valor de la máscara en hexadecimal.

COMENTARIO FINAL

Obviando que se ha desarrollado sobre un simulador y no sobre un procesador real, los límites de MiMoS más significativos son los siguientes:

- PCSpim no simula el modo monitor del procesador, de manera que el programa usuario puede, por ejemplo, ejecutar instrucciones privilegiadas y acceder directamente a la interfaz de los periféricos sin necesidad de invocar a MiMoS.

- PCSpim no simula la unidad de gestión de memoria virtual del MIPS y por eso MiMoS no puede implementar mecanismos de protección de memoria.

- MiMoS está escrito en ensamblador, así que resultaría muy difícil dotarlo de muchos rasgos propios de los actuales sistemas operativos que exigen programación minuciosa y estructuras de datos complejas.

Con semejantes límites, se han escrito sistemas operativos relevantes. El CP-M y el DOS fueron dos sistemas operativos para microprocesador que fueron desarrollados con métodos parecidos al de esta práctica. Naturalmente, el trabajo se hizo en equipo y utilizando herramientas especiales de desarrollo.