

Práctica 14: Llamadas al sistema operativo (I)

Introducción y objetivos

El juego de instrucciones del MIPS ofrece primitivas para implementar un sistema operativo: modos de funcionamiento, excepciones, organización del coprocesador y otros. Para practicar con los métodos básicos de entrada/salida, se ha definido un sistema operativo rudimentario llamado MiMoS (*Mips Monitor System*). La estructura de este manejador es semejante a la del manejador que se completó en la práctica anterior. En aquel manejador solo se tenía la capacidad de manejar las interrupciones procedentes de las líneas INT0*, INT1* e INT2*. Este manejador además es capaz de gestionar la llamada al sistema syscall. Y se le ha dotado de recursos para el soporte multiproceso.

Los objetivos de esta práctica son:

- ☐ Implementar nuevas funciones accesibles como llamadas al sistema operativo (syscall). Inicialmente se proporciona un manejador *MIMOSv0.handler*, donde se encuentran ya implementadas dos llamadas al sistema *get_version*, y *print_char*.
- ☐ Comprender el funcionamiento multiproceso de los sistemas operativos, implementando un sencillo mecanismo de gestión de procesos. Mediante una llamada al sistema se dejará en espera el proceso actual y se pasará a ejecutar un proceso ocioso. Una interrupción volverá a poner en ejecución el proceso en espera.

Material

- ☐ La versión del simulador PCSIM_ES utilizada en la práctica anterior.
- ☐ La versión preliminar del manejador *MIMOSv0.handler*.
- ☐ Archivos de prueba: *Usuario0.s*, *Usuario1.s*, *Usuarios2.s* y *Usuario3.s*. Estos archivos servirán para poner a prueba las funciones de los distintos manejadores de excepciones que se irán desarrollando a lo largo de la práctica. *Usuario0.s* realiza llamadas a las funciones definidas en el manejador *MIMOSv0.handler*. *Usuario1.s* al manejador *MIVOSv1.handler*, y así sucesivamente.
- ☐ Tratamiento de excepciones con el simulador PCSpim.
- ☐ Apéndices: llamadas al sistema, registro estado MIPS, interfazTecladoConsolaReloj.

El simulador ha de estar configurado para que las interrupciones y las llamada syscall provoquen la ejecución del manejador de excepciones, como muestra la Figura 1.

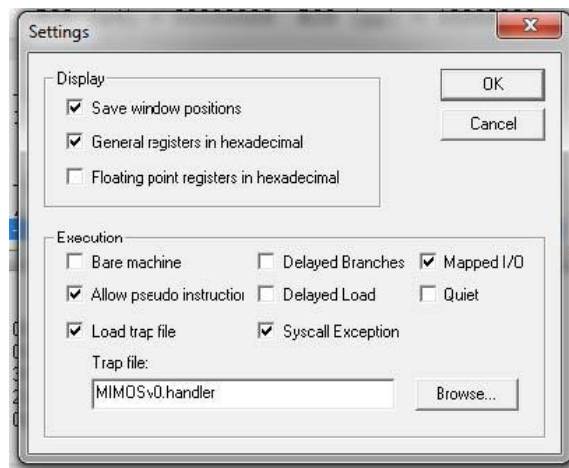


Figura 1: Configuración del simulador para que un manejador llamado *Mimosv0.handler* atienda las llamadas al sistema.

Recuerda los detalles vistos en la práctica anterior: A lo largo de esta práctica, se trabajará con dos archivos:

- ☐ El manejador o *Trap file*, con la extensión *.handler*, define los segmentos *.kdata* y *.ktext* del manejador de excepciones y un fragmento del segmento *.text* para el código de inicio y terminación que haga falta.
- ☐ El programa de usuario, con extensión *.s*, describe el resto de los segmentos *.data* y *.text*.

Cada vez que se abra (*File>Open*) o recargue (*Simulator>Reload*) un archivo de usuario, el simulador cargará también el archivo manejador etiquetado como *Trap File* en el cuadro de configuración de la figura 1.

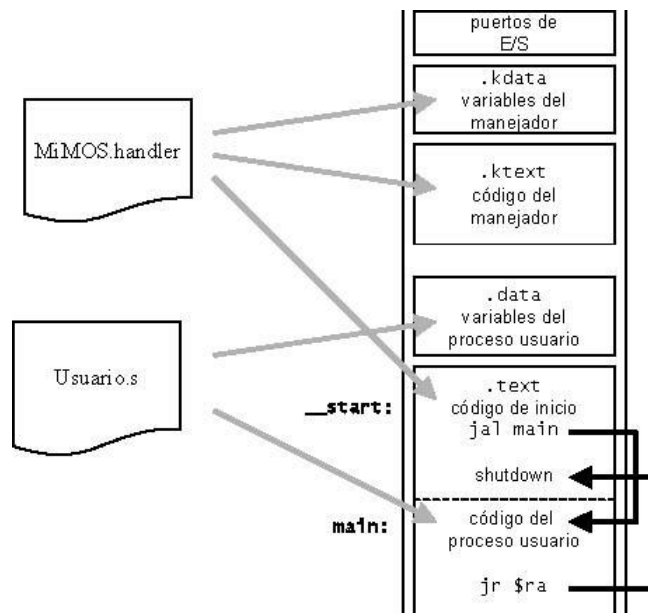


Figura 2: Los dos archivos fuente que describen la memoria del MIPS en una sesión MiMoS.

Recuerde también que el inicio del sistema supone tres pasos:

1. Preparar los periféricos disponibles, habilitando o inhibiendo las interrupciones a través de sus respectivas interfaces
2. Preparar el registro de estado del coprocesador de excepciones: máscara de interrupciones y modo de funcionamiento del procesador
3. Transferir el control al programa usuario.

El paso 3 se hace mediante `jal main`, así que el programa de usuario debe tener un punto de entrada etiquetado como `main`. Para acabar su ejecución, el programa de usuario debe ejecutar `jr $ra` (Figura 2).

Estructura del manejador MiMoS

En la Figura 3 aparece la descomposición en bloques del manejador, aparecen en el mismo orden que en el archivo fuente *MiMoSv0.handler* y con las mismas etiquetas. Solo se ha añadido el tratamiento de la *syscall*. Solo hay dos llamadas al sistema implementadas *print_char* y *get_version*. A lo largo de la práctica se irán añadiendo nuevas funciones.

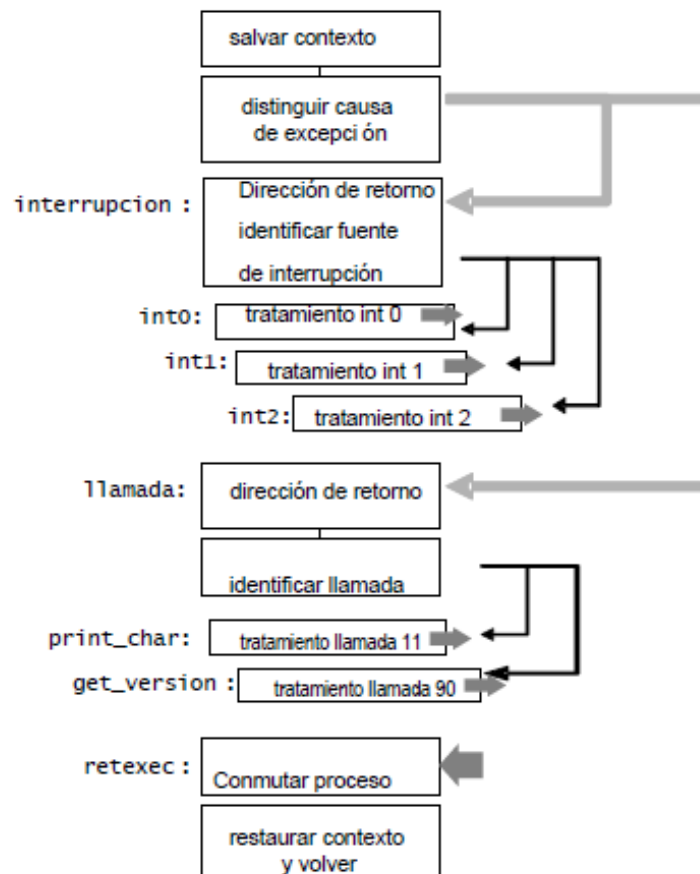


Figura 3: Estructura del manejador MiMoS. Los símbolos `·` representan saltos a la etiqueta `retexc`.

El manejador, después de salvar el contexto, sólo distingue como causa de excepción la interrupción de periférico y la llamada al sistema. En ambos casos, y después de almacenar la dirección de retorno, hay una sección de identificación (línea de interrupción o número de función, según el caso) que salta a la etiqueta donde comienza el tratamiento asignado. Cada tratamiento debe acabar con la instrucción `b retexc` para saltar a la sección de conmutación y restauración de contexto.

La dirección de retorno se almacena de un modo diferente a como se ha visto en teoría por simplicidad a la hora de elegir entre proceso ocioso y principal. Solo cuando el proceso interrumpido es el principal se guarda la dirección de retorno, si se tratara del proceso ocioso la dirección de retorno es conocida, ya que solo tiene una instrucción.

En *MiMoSv0.handler*, todos los bloques de tratamiento de interrupción están por completar, y las interrupciones generales están habilitadas, pero tanto lo periféricos como las líneas están enmascaradas. Las funciones que tratará la syscall (excepto *get_version* y *print_char*) están por desarrollar, aunque las etiquetas y el salto final del tratamiento ya están definidos. El resto de secciones está ya elaborado y no se tendrá que modificar en esta práctica.

Tarea 1. Manejador versión 0 (MiMoS v.0). Funciones *get_version* y *print_char*.

El manejador *MiMoSv0.handler* implementa la versión 0 de MiMoS. Sólo tiene implementados dos servicios: *get_version* y *print_char* (ver tabla 1).

Función	Código	Argumentos	Resultados
<i>get_version</i>	$\$v0 = 90$		$\$v0$ = número de versión
<i>print_char</i>	$\$v0 = 11$	$\$a0$ = carácter	Imprimir carácter en consola

Tabla 1: Servicios implementados en *MiMoS v.0*

► Observe el código del manejador de MiMoS v.0 y compruebe que se ajusta a la estructura de la Figura 3. Busque la implementación de los dos servicios, *get_version* sólo devuelve $\$v0 = 0$ y *print_char* escribe un carácter en la consola sincronizándose por consulta de estado. A lo largo de la práctica se deberá completar el código del manejador para obtener nuevas versiones; y en cada fase se tendrá que modificar el tratamiento de la función *get_version*.

► Observe también el código de inicio del sistema. Como *MiMoS v.0* no atiende ninguna interrupción, fíjese en que cada una de las interrupciones previstas (reloj, teclado y consola) está deshabilitada por partida doble: en su interfaz (en archivo *interfazTecladoConsolaReloj* está la descripción) y en el registro de estado del coprocesador de excepciones (en archivo *registro estado MIPS*).

► Para probar el manejador es necesario un programa de usuario que utilice las funciones que suministra. Utilice *Usuario0.s* para probar *MiMoSv0.handler*. Este programa utiliza las funciones de la Tabla 1 para escribir el siguiente texto en la consola:

```
MiMoS v.0
1
2
...
```

► Pruebe el sistema completo en el simulador. No olvide ajustar el cuadro de configuración antes de cargar el código de prueba (Figura 1). Puede detener la simulación cuando quiera pinchando en la barra de menús de *PCSpim* la orden *Simulator>Break* o tecleando [*Control-C*].

Cuestión 1. Teniendo *Usuario0.s* en ejecución, detenga el simulador. (Antes de responder sí en el cuadro “Execution paused by the user at PC = ...”, asegúrese que el *PC* apunta a una instrucción del programa de usuario, cuyas direcciones son del tipo “0x0040nnnn”. Si apunta a direcciones del tipo “0x8000nnnn” pulse que no y vuelva a intentarlo).

► ¿Qué vale el registro Status (registro \$12) que aparece en la ventana superior del simulador?

Status = 00000003, que mirant el apendice 2 podem veure que significa que estan habilitades les interrupcions i esta en mode usuari.

► ¿En qué modo está funcionando el procesador? ¿Están habilitadas las interrupciones?

Mirant el apendice 2 veiem que estem en mode usuari i les interrupcions estan habilitades

► ¿Qué valen los bits de máscara de interrupción?

00000003 segons el apendice 2--> el valor seria 00

► Indique con qué instrucciones del código de inicio de se *MiMoSv0.handler* se inhiben las interrupciones en los periféricos teclado, reloj y consola.

```
## Preparo las interfaces de los periféricos
li $t0, 0xffff0000
sb $zero, 0($t0)    # inhibo interrupción en el HW del teclado

li $t0, 0xffff0008
sb $zero, 0($t0)    # inhibo interrupción en el HW de la consola

li $t0, 0xffff0010
sb $zero, 0($t0)    # inhibo interrupción en el HW del reloj
```

► Indique cómo se inicializa el registro de estado del coprocesador 0.

```
## Preparo el registro de estado del coprocesador y fijo modo usuario
mfc0 $t0, $12
ori $t0, $t0, 0x0003 # Interrupciones habilitadas
mtc0 $t0, $12

carreguem un 3 per a colocar els bits IE i KU del apendice 2 a 1, aixina habilitem el mode usuari i les interrupcions
```

A partir de ahora, se han de crear distintas versiones de *MiMoS* en las que sucesivamente se acumularán nuevas funciones. **Cree siempre cada versión a partir de la anterior.**

Tarea 2. Manejador versión 1 (MiMoS v.1). Función `get_time`.

Se ha de dotar a *MiMoS* de los servicios mostrados en la Tabla 2. Es decir, además de los dos servicios implementados en la tarea anterior, el sistema medirá el paso del tiempo con una resolución de 1 segundo, y con la nueva función `get_time` se podrán obtener los segundos.

Función	Código	Argumentos	Resultado
get_version	$\$v0 = 90$		$\$v0$ = número de versión
print_char	$\$v0 = 11$	$\$a0$ = carácter	Escribe carácter en consola
get_time	$\$v0 = 91$		$\$v0$ = Tiempo en segundos

Tabla 2: Servicios implementados en MiMoS v.1

- Cree una copia del archivo *MiMoSv0.handler* llamada *MiMoSv1.handler* y edite este último. Modifique el servicio `get_version` para que devuelva el valor 1.
- En la sección de variables del manejador (zona indicada como “Variables para el reloj”) se ha de añadir una variable de tipo *word* de nombre *segundos* con valor inicial 0.
- En la zona de tratamiento de interrupciones, se ha de insertar, a partir de la etiqueta `int2`, el tratamiento de la interrupción de reloj que se producirá cada segundo. El tratamiento adecuado será cancelar la interrupción poniendo a 0 el bit EOC de la interfaz del reloj (descrita en el archivo *interfazTecladoConsolaReloj*) e incrementar en 1 la variable *segundos*.

```
int2: segundos = segundos + 1;
      cancelar interrupción;
      b retexc
```

Cuestión 2. Escriba el código de tratamiento de la interrupción del reloj.

```
int2:  # Por implementar
        la $t0, segundos
        lw $t1, 0($t0)
        addi t1, t1, 1
        sw $t1, 0($t0)

        li $t0, 0xffff0010
        li $t1, 1
        sb $t1 0($t0)

        b retexc          # fin
```

Cuestión 3. Modifique el código de inicio del sistema para que la interrupción del reloj quede habilitada, tanto en la interfaz como en el registro de estado del coprocesador 0.

```
## Preparo las interfaces de los periféricos
li $t0, 0xffff0010
li $a0, 1
sb $a0 0($t0)      # habilite interrupción en el HW del reloj

## Preparo el registro de estado del coprocesador y fijo modo usuario
mfc0 $t0, $12
ori $t0, $t0, 0x0403 # Interrupciones habilitadas
mtc0 $t0, $12
```

► A continuación se ha de añadir el código apropiado en la zona de llamadas al sistema a partir de la etiqueta `get_time`. El tratamiento es muy parecido a `get_version`, pero en vez de cargar en `$v0` una constante se ha de copiar el valor de la variable *segundos*.

```
get_time:
    $v0 = segundos;
    b retexc
```

Cuestión 4. Escriba el código de la función *get_time* a continuación.

```
get_time:
    lw $v0, segundos
    b retexc
```

► Actualice el cuadro de configuración de PCSpim para que cargue el nuevo manejador y compruebe la nueva función utilizando el archivo *Usuario1.s*, que hace lo siguiente:

```
Saludo inicial;
Repetir
    calcular
    llamar a get_time
    escribir tiempo actual
```

Cuestión 5. ¿Podría ejecutarse correctamente *Usuario0.s* con el manejador *MiMoSv.1*? ¿Podría ejecutarse correctamente *Usuario1.s* con el manejador *MiMoS v.0*? Razone la resupuesta.

Usuario0.s con MIMOSv1.handle si que funciona debido a que Usuario0.s no hace ninguna llamada al sistema. Sin embargo al revés no funciona debido a que las excepciones están deshabilitadas en el manejador 0 y el Usuario1.s usa interrupciones.

Los procesos en MiMoS

Podemos abstraer el programa contenido en el archivo de usuario y denominarlo **proceso principal**. Hasta ahora, este proceso se encontraba siempre activo, pero en adelante se tendrán que utilizar otros estados. El **estado** del proceso principal se encuentra almacenado en la variable estado del manejador. Hay definidos (como constantes) dos estados.

```
## Estados del proceso principal
LISTO = 0
ESPERANDO = 1

estado: .word LISTO # Estado del proceso
```

También hay definido un proceso ocioso, al que se conmuta cada vez que el proceso principal no está listo. El código del proceso ocioso es 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, debe determinar al final de su ejecución a qué instrucción debe retornarse. 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
```

El **cambio de contexto** no es necesario en este caso, porque sólo hay que mantener el contexto del proceso principal, dado que el proceso ocioso no requiere salvar ningún contexto (no utiliza registros y la dirección de retorno es siempre la misma y la única del proceso). Durante la ejecución del manejador, además del valor del PC de retorno, que se guarda en la variable *dirret*, se preservan tres registros de propósito general (*\$at*, *\$t0* y *\$t1*) en la variable *salvareg* para que pueda utilizarlos el código del manejador. Si se necesitan más registros para los tratamientos, se puede ampliar *salvareg* y modificar el código de salvar contexto (al principio del manejador) y restaurar contexto (al final) para que se preserven más registros.

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

En esta tarea se va a dotar al manejador MiMoS de la función *wait_time* mostrada en la Tabla 3. El servicio permite al proceso principal esperar el número de segundos especificado como argumento

Función	Código	Argumentos	Resultado
wait_time	<i>\$v0</i> = 92	<i>\$a0</i> = tiempo en segundos	Proceso en espera <i>\$a0</i> seg

Tabla: Nuevo servicio de espera que se ha de añadir en MiMoS v.2

La función *wait_time* es un ejemplo típico de funciones del sistema que producen esperas temporales. Para resolver esta cuestión vamos a utilizar la gestión de procesos indicada anteriormente. La idea consiste en cambiar el estado del proceso que llama a la función desde el valor LISTO al valor ESPERANDO. Con esta acción se hará que el manejador no retorne al proceso principal, sino que retornará al proceso ocioso. La rutina de interrupción del reloj será la encargada de resaturar el estado al valor LISTO trascurridos los segundos indicados en la función *wait_time*. El proceso se esquematiza en la figura 4 .

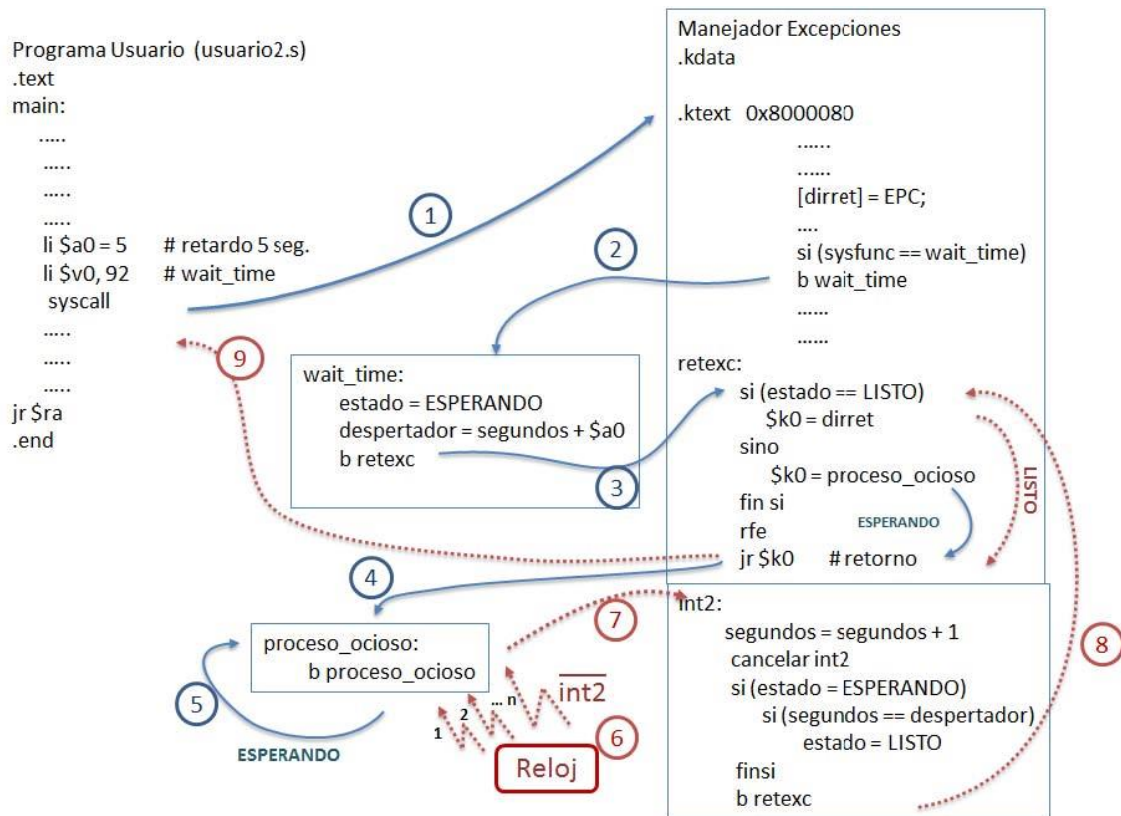


Figura 4 – Conmutación de procesos en la llamada a *wait_time*.

Los eventos que se suceden durante la espera son los siguientes:

1. Llamada a la función del sistema con número 92 (*wait_time*).
2. El manejador de excepciones guarda la dirección de retorno del proceso principal en *dirret* y comprueba que es la función 92 y salta a la rutina *wait_time*.
3. La rutina *wait_time* cambia el estado del proceso principal al valor ESPERANDO y salta a *retexc* para retornar.
4. El código de retorno comprueba que el proceso NO está en estado LISTO y, por lo tanto, retorna al proceso ocioso.
5. El proceso ocioso queda en un bucle de espera muerta (no hace nada).
6. Se reciben varias interrupciones del reloj. Cada interrupción incrementa la variable *segundos*.
7. En la *n*-ésima interrupción del reloj (*n* = despertador) se comprueba que la espera debe acabar, para lo cual se cambia el estado del proceso al valor LISTO y se salta a *retexc* para retornar.
8. El código de retorno comprueba que el proceso SI está en estado LISTO y, por lo tanto, la dirección de retorno será *dirret*.
9. Retorna al proceso principal que estaba esperando.

► Cree una copia del archivo *MiMoSv1.handler* llamada *MiMoSv2.handler* y edite este último. Modifique el servicio `get_version` para que devuelva el valor 2.

► El tratamiento de la llamada `wait_time` consistirá en dejar el proceso principal en estado ESPERANDO y calcular y almacenar en una nueva variable despertador en qué momento (valor actual de segundos más el tiempo de espera especificado) se reactivará el proceso.

```
wait_time:
    estado = ESPERANDO;
    despertador = segundos + $a0;
    b retexc
```

Cuestión 6. Escriba el código de la función *wait_time* que se ha implementado.

```
wait_time:  la $t0, estado
            li, $t1, ESPERANDO
            sw $t1, 0($t0)

            la $t0, segundos
            lw $t1, 0($t0)

            add $t1, $t1, $a0
            la $t0, despertador
            sw $t1, 0($t0)

            b retexc
```

► El tratamiento del reloj (en la etiqueta `int2`), después de incrementar la variable `segundos`, ha de comprobar si el estado del proceso es ESPERANDO. En este caso, si `segundos = despertador`, devolver el proceso principal a estado LISTO.

```
int2:
    segundos = segundos + 1;
    cancelar interrupción;
    si (estado = ESPERANDO)
        si (segundos = despertador) estado =
            LISTO;
        finsi
    finsi
    b retexc
```

Cuestión 7. Escriba el código de la interrupción del reloj, desde la etiqueta int2.

```
int2:      la $t0, segundos
          lw $t1, 0($t0)
          addi $t1, $t1, 1
          sw $t1, 0($t0)

          li $t0, 0xffff0010
          li $t1, 1
          sb $t1, 0($t0)

          la $t0, estado
          lw $t1, 0($t0)
          li $t2, ESPERANDO

          bne $t1, $t2, fin
          la $t0, segundos
          lw $t1, 0($t0)
          la $t0, despertador
          lw $t2, 0($t0)

          bne $t1, $t2, fin
          la $t0, estado
          li $t1, LISTO
          sw $t1, 0($t0)

          fin:      b retexc      # fin
```

► Pruebe el nuevo manejador, actualice el cuadro de configuración de PCSpim y carga el programa *usuario2.s*. Es un programa de prueba que sigue el siguiente esquema:

Saludo inicial;

Repetir

Llamada a get_time;

escribir tiempo actual;

Llamada a wait_time (5 segundos);

Cuestión 8. Detenga el programa *usuario2.s* justamente después de haber escrito el tiempo actual (**n segundos**). ¿Qué valen el PC y el registro **Status**?

403

¿Qué código se está ejecutando: el manejador, el proceso de usuario o el proceso ocioso?

El proceso ocioso