SISTEMAS OPERATIVOS

CURSO 2023-2024

# SIMULADOR DE UN SISTEMA INFORMÁTICO MULTIPROGRAMADO CON INTERRUPCIONES DE RELOJ

# V2

## Introducción

A partir del conocimiento adquirido con el SI en las versiones V0 y V1, continuamos nuestro estudio de los SO con una evolución del Sistema Informático al que habremos dotado de gestión de interrupciones de reloj, y del uso del estado BLOCKED de procesos, que antes no se usaba (cuando hayamos completado los ejercicios de esta versión).

Se han producido en el simulador cambios en sus diferentes componentes, que pasamos a detallar. Donde no haya habido cambios o estos hayan sido poco relevantes, no se detallará la naturaleza de los mismos.

## DISEÑO

### *El sistema operativo*

- Estructuras de datos añadidas/modificadas:
  - Contador de interrupciones: `numberOfClockInterrupts`
  - Tabla de procesos: aparece nueva información.
  - Cola de procesos dormidos: `sleepingProcessesQueue` gestionada como un montículo binario, usando el número de interrupción de reloj en la que debe despertar el proceso como valor de ordenación. La gestión se hace de forma similar a la `readyToRunQueue`
- Funcionalidad:
  - Planificador a Corto Plazo (PCP) o Short-Term Scheduler (STS).
    - Mantiene de la V1 el criterio de prioridad de los procesos de usuario frente a los de sistema.
  - Rutinas de tratamiento de interrupción.
    - Aparece una nueva rutina de tratamiento de las interrupciones de reloj. Por tanto, el SO debe implementar una rutina de tratamiento de interrupción para cada tipo de interrupción reconocida por el procesador:
      - Para las excepciones.
      - Para las llamadas al sistema (aparece una nueva llamada que se suma a la aparecida en la V1).
      - Para las interrupciones de reloj.

### *El procesador*

Aparece la posibilidad de enmascarar el tratamiento de interrupciones nuevas cuando se está tratando una interrupción o el sistema se está apagando.

# SIMULADOR DE UN SISTEMA INFORMÁTICO MULTIPROGRAMADO CON INTERRUPCIONES DE RELOJ

# TAREAS V2

## Tareas iniciales

Saca un duplicado de tu directorio `V1` (con todos los ejercicios de la V1 terminados) denominándolo `V2`.

A continuación, descarga el código del Campus Virtual y copia el contenido en tu directorio `V2`.

Finalmente, ya dentro de tu directorio `v2`, ejecuta la siguiente orden:

```
$ make clean
```

El trabajo para realizar en los ejercicios siguientes se desarrollará sobre la copia indicada de los ficheros contenidos en el directorio `V2`, dentro de tu directorio personal.

## Ejercicios

1. Vamos a introducir una nueva interrupción al sistema, para lo cual se define entre otras cosas, un nuevo manejador de interrupciones.

    a. Establece el bit 9 de la `interruptLines_CPU` para que corresponda a la interrupción de reloj (`CLOCKINT_BIT=9`) en el enumerado INT_BITS de `Processor.h`.

    b. Pega el código de debajo en el fichero indicado en el comentario correspondiente y añade la función prototipo donde sea necesario.

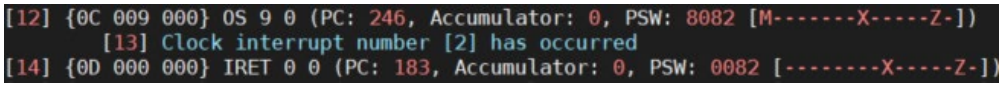
    ```
    // In OperatingSystem.c  Exercise 1-b of V2
    void OperatingSystem_HandleClockInterrupt(){ return; }
    ```

    c. Modifica el **sistema** para que la función que trate las interrupciones de reloj introducidas en el ejercicio anterior pase a ser: `OperatingSystem_HandleClockInterrupt()`. Comprueba cómo se hace con otras interrupciones para ver cómo hacerlo.

    d. Modifica la función del reloj del sistema `Clock_Update()` para que produzca una interrupción de reloj cada `intervalBetweenInterrupts` unidades de tiempo una vez incrementado el tic de reloj. Siendo `intervalBetweenInterrupts` una variable del sistema que por defecto tiene valor 5, aunque puede pasarse su valor al simulador usando la opción: `--intervalBetweenInterrupts`.

    e. Modifica la rutina `OperatingSystem_HandleClockInterrupt()` para que cuente el número total de interrupciones de reloj ocurridas (en la variable `numberOfClockInterrupts`) y muestre un mensaje con tiempo en pantalla con el

aspecto siguiente (número de mensaje 120, sección `INTERRUPT`, y color Cyan); en el que el número se corresponde con la interrupción de reloj que se está tratando:




2. Piensa en las situaciones que se pueden dar al solaparse las nuevas interrupciones de reloj con las interrupciones existentes.

   Vamos a modificar el procesador para permitir el enmascaramiento de interrupciones si está activado determinado bit de la PSW.

   a. Añade el valor `INTERRUPT_MASKED_BIT=15` al enumerado `PSW_BITS` en `Processor.h`

   b. Para que muestre en la PSW el nuevo bit (en la representación con letras de los bits), hay que modificar la función `Processor_ShowPSW()` que es la que la muestra.

      Para hacerlo, añade antes del `return` de la función `Processor_ShowPSW()` lo siguiente:

      ```
      if (Processor_PSW_BitState(INTERRUPT_MASKED_BIT))
            pswmask[tam-INTERRUPT_MASKED_BIT]='M';
      ```

   c. Modifica la función `Processor_ManageInterrupts()` para que no pase a comprobar las posibles interrupciones activas, en el caso de que estén enmascaradas las interrupciones.

   d. Activa el `INTERRUPT_MASKED_BIT` de la PSW una vez apilada la PSW del proceso interrumpido en el manejador de interrupciones.

3. Añadir una llamada a la función: `OperatingSystem_PrintStatus()` que muestra el estado del simulador (podéis verla en `OperatingSystemBase.c`).
   a. Como última instrucción del `OperatingSystem_Initialize()`.
   b. Como última instrucción del tratamiento de la llamada al sistema SYSCALL_YIELD **SOLO** en el caso de que haya cambiado el proceso en ejecución.
   c. Como última instrucción del tratamiento de la llamada al sistema SYSCALL_END.
   d. Como última instrucción del manejador de excepciones.
   e. Al final del planificador a largo plazo, **SOLO** en el caso de que se haya creado algún proceso en la llamada al PLP.

4. A estas alturas, ya deberían funcionar correctamente las colas de listos, y ya no es necesario mostrar las colas de listos cada vez que se mete un proceso en ellas. Además, ya se hace en `OperatingSystem_PrintStatus()`; así que para evitar mensajes confusos y/o repetidos, comentad la llamada a `OperatingSystem_PrintReadyToRunQueue()` al final de la función `OperatingSystem_MoveToTheREADYState()` introducida en el ejercicio 9-b de la V1.

5. Vamos a introducir una nueva llamada al sistema:

   a. Añade al PCB un campo adicional:

      ```
      int whenToWakeUp;  // Exercise 5-a of V2
      ```

   b. Pega este código en el fichero indicado:

      ```
      // In OperatingSystem.c  Exercise 5-b of V2
      // Heap with blocked processes sort by when to wakeup
      heapItem *sleepingProcessesQueue;
      int numberOfSleepingProcesses=0;
      ```

      Y crea una cola de prioridad (`Heap`) para tantos procesos como pueda soportar el sistema (`PROCESSTABLEMAXSIZE`) de forma similar a como se reserva memoria para las colas de listos.

   c. Define `SLEEPINGQUEUE` en `OperatingSystem.h` para que compile el código que depende de que se hayan definido las estructuras de la cola de procesos bloqueados.

      ```
      #define SLEEPINGQUEUE
      ```

   d. Define un nuevo registro en el procesador llamado `registerD_CPU`; y haz la función que permita acceder a su valor.

   e. Modifica la instrucción TRAP, para que guarde en el nuevo registro (`registerD_CPU`), el operando 2.

   f. Añade una nueva llamada al sistema `SYSCALL_SLEEP=7` que bloqueará al proceso en ejecución (es decir, se tendrá que mover al estado BLOCKED) y lo insertará por orden creciente del campo `whenToWakeUp` en la `sleepingProcessesQueue`. Como todas las llamadas a sistema, se invocará con la instrucción TRAP, pero se usará el segundo operando para pasar un valor adicional a la llamada a sistema.

      El valor del campo `whenToWakeUp` se obtendrá sumando: un "*retardo*" al número de interrupciones de reloj que se hayan producido hasta el momento; más una unidad adicional, para garantizar que se tiene que despertar en el futuro: *whenToWakeUp=retardo+numInterrup+1*.

      Si el segundo operando es mayor estricto de cero, se usa su valor como "*retardo*", y si es menor o igual que cero, se usa como "*retardo*" el valor absoluto del acumulador.

      Es decir, si el valor del segundo operando es 2 y se han producido ya 3 interrupciones de reloj, `whenToWakeUp` valdrá 2+3+1=6; lo mismo que si el segundo operando fuese <=0, y en el acumulador hubiese un 2 o un -2: Abs(±2)+3+1.

      Se **recomienda** crear una función para meter en la cola de procesos bloqueados `sleepingProcessesQueue` de forma similar a como se hace en la cola de procesos listos para ejecución `readyToRunQueue`

   g. Añadir como **última instrucción** del tratamiento de la llamada al sistema, una llamada a la función: `OperatingSystem_PrintStatus()` que muestra el estado del simulador.

En este punto, los procesos son capaces de dormirse (pasar a estado BLOCKED), pero no hay forma de despertarlos, por lo que es posible que se generen bucles infinitos en el simulador.

6. Modifica la función `OperatingSystem_HandleClockInterrupt()` para que el sistema operativo revise la `sleepingProcessesQueue` cada vez que se produce una interrupción de reloj, de la manera siguiente:

   a. Si el campo `whenToWakeUp` de un proceso (**o más de uno**) de la cola `sleepingProcessesQueue` coincide con el número de interrupciones de reloj ocurridas hasta el momento, el proceso se desbloqueará, pasando al estado READY y siendo eliminado de la `sleepingProcessesQueue`.

      Se **recomienda** crear una función para sacar de la cola de procesos bloqueados `sleepingProcessesQueue` de forma similar a como se hace en la cola de procesos listos para ejecución `readyToRunQueue`

   b. Una vez procesada la `sleepingProcessesQueue`**, si se ha desbloqueado algún proceso**, se deberá mostrar el nuevo estado del sistema mediante una llamada a la función: `OperatingSystem_PrintStatus()`

   c. Además de lo indicado en el punto anterior, será necesario **comprobar si el proceso en ejecución sigue siendo el más prioritario** de todos los que pueden competir por el procesador. En caso de no serlo, será necesario **sustituir al proceso en ejecución por el proceso más prioritario**, mostrando además un mensaje con tiempo con el aspecto siguiente (número de mensaje 121, sección `SHORTTERMSCHEDULE`):

```
[27] Process_[1_-_prNam1]_will_be_thrown_out_of_the_processor_by_process_[2_-_prNam2]
```

   d. Añadir como última instrucción, en el caso de que se haya cambiado el proceso en ejecución, una llamada a la función: `OperatingSystem_PrintStatus()`

Una vez terminados todos los ejercicios de la V2, debería mostrarse el estado del sistema por el uso de la función `OperatingSystem_PrintStatus()`:

- Cada vez que el planificador a largo plazo cree procesos nuevos.
- Cada vez que se despierta/n algún/os proceso/s.
- Cada vez que se cambia el proceso en ejecución.

MainMemory

MAR

MBR

mainMemory

0 1 2 3 ⋮ 59 60 61 62 ⋮ 119 120 ⋮ 179 180 ⋮ 239 240 ⋮ (*)

(*) = MAXMEMORYSIZE - 1

MMU

Base

Limit

MAR

ComputerSystem

UserProgramList

| | *PROGRAMDATA | *Name | Arrival | Type |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | null | | | |
| | | | | |
| (*) | null | | | |

(*) =PROGRAMMAXNUMBER-1

debugLevel

DebugMessages

| | | |
|---|---|---|
| | … | |
| 68 | 68 | {%s} |
| 69 | 69 | %s %d %d (PC: @R%d@@, Accumulator: @R%d@@), PSW: @R%x@@ [@R%s@@])\n |
| | … | … |
| (*) | | |

(*) NUMBEROFMSGS -1

Clock

tics

Processor

accum

PC

IR

MAR

MBR

PSW

A

B

C

D

SP

Int

VInt.

| | |
|---|---|
| 0 | void |
| 1 | void |
| 2 | SysCall Entry |
| 3 | void |
| 4 | void |
| 5 | void |
| 6 | Exception Entry |
| 7 | void |
| 8 | void |
| 9 | ClockInt Entry |
| … | … |
| (*) | void |

(*) = INTERRUPTTYPES -1

OperatingSystem

executingProcessID

sipID

NonTerminated

numberOfClockInterrupts

ProcessTable

| | PID | busy | initialAddr | size | state | priority | Copy PC | queueID | whenToWakeUp | … |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | |
| 1 | | | | | | | | | | |
| 2 | | | | | | | | | | |
| … | | | | | | | | | | |
| (*) | | | | | | | | | | |

ReadyToRun

| | 0 | 1 | 2 | … | (*) |
|---|---|---|---|---|---|
| USER | | | | | |
| DAEM | | | | | |

NumReadyToRun

| | |
|---|---|
| USER | |
| DAEM | |

Sleeping

| 0 | 1 | 2 | … | (*) |
|---|---|---|---|---|
| | | | | |

NumSleeping

(*) PROCESSTABLEMAXSIZE - 1

**Nota**: se han omitido del esquema los Buses, pero se siguen utilizando.