SISTEMAS OPERATIVOS

SIMULADOR DE UN SISTEMA INFORMÁTICO MULTIPROGRAMADO V1 (y sucesivas versiones)

COMPROBACIÓN DE ASERTOS

Introducción

Para verificar de una manera sencilla el funcionamiento del simulador en la ejecución de un programa determinado, se ha desarrollado un sistema de verificación del contenido de diversos elementos del simulador en determinados instantes de tiempo.

Así, el usuario puede definir un fichero (denominado por defecto *asserts*) donde introducirá el valor esperado de determinados elementos del sistema en ciertos instantes de tiempo. En cada uno de esos instantes de tiempo se verificarán que los valores esperados se corresponden con los previstos, dando un mensaje de error en otro caso.

Modificaciones al simulador introducidas

- Añadido variable GEN_ASSERTS. Es un *flag* que indica si se utiliza un fichero para generar un fichero de asertos con los valores de las variables en los momentos indicados, o si se comprueban los asertos. Si vale 1, se genera, si vale 0 se comprueba. Su valor se establece a 1 si se incluye la opción `--generateAsserts` en los parámetros de la llamada al Simulador.
- Se utiliza el Clock (`Clock.c` y `Clock.h`) para saber el tiempo de cada aserto.
- `Asserts.c` y `Asserts.h`. Incorporan la lógica para verificar los asertos.
- `MyAspect.c`. Incluye la llamada a `Assert_CheckAsserts` después de la ejecución de `Processor_DecodeAndExecuteInstruction`.
- `asserts`: O el fichero indicado en la opción `--assertsFile`, contiene los asertos a ser verificados durante la ejecución del programa. Si no existe, no se comprueba nada (da un mensaje al principio indicando que no se encuentra el archivo).

Funcionamiento general

Al principio de la ejecución del simulador, se cargan los asertos en memoria.

Si no hay fichero de asertos no se comprueban asertos.

Justo después de la ejecución de `Processor_DecodeAndExecuteInstruction` se verifica si hay algún aserto que comprobar para ese instante de tiempo. Si es así, se comprueba que el elemento que sea tenga el valor esperado. Si ese valor no es el esperado, muestra un mensaje por pantalla.

Los mensajes se muestran usando `ComputerSystem_DebugMessage` en la sección ERROR, por lo que siempre salen.

Para verificar si el simulador cumple con los asertos definidos se recomienda utilizar el simulador sin mensajes de depuración (opción “n”).

Si se utiliza la opción `--generateAsserts` se generan por pantalla las líneas que podrán copiarse al fichero de asertos, como asertos a comprobar; en vez de comprobar los asertos en la ejecución.

Asertos

Los asertos deben estar almacenados en un fichero (`asserts`. por defecto), y tienen la siguiente forma:

```
instante, elemento, valor [,direccion]
```

Cada aserto puede leerse como “tras el **`instante`** indicado el **`elemento`** señalado contendrá el **`valor`** que se muestra”. En caso de que el elemento sea una posición de memoria, un cuarto parámetro indicará la **`dirección`** de esa posición de memoria.

El valor podrá ser una cadena de caracteres (si se está comprobando un código de operación), o un valor numérico, dependiendo del elemento que se esté consultado.

Los elementos que se pueden consultar son los siguientes (se pueden consultar en `Asserts.h`):

- RMEM_OP: Código de operación almacenado en la posición de memoria relativa del proceso en ejecución indicada:
  `5, RMEM_OP, ADD, 3` (En el instante 5 el código de operación de la dirección 3 es ADD_INST).
- RMEM_O1: Operando 1 almacenado en la posición de memoria relativa del proceso en ejecución indicada:
  `5, RMEM_O1, 132, 3` (En el instante 5 el operando 1 que hay en la dirección 3 es 132).
- RMEM_O2: Operando 2 almacenado en la posición de memoria relativa del proceso en ejecución indicada:
  `5, RMEM_O2, 13, 3` (En el instante 5 el operando 2 que hay en la dirección 3 es 13).
- AMEM_OP: Código de operación almacenado en la posición de memoria absoluta indicada:
  `5, AMEM_OP, ADD, 140` (En el instante 5 el código de operación de la dirección absoluta 140 es ADD_INST).
- AMEM_O1: Operando 1 almacenado en la posición de memoria absoluta indicada:
  `5, AMEM_O1, 132, 140` (En el instante 5 el operando 1 que hay en la dirección absoluta 140 es 132).
- AMEM_O2: Operando 2 almacenado en la posición de memoria absoluta indicada:
  `5, AMEM_O2, 13, 140` (En el instante 5 el operando 2 que hay en la dirección absoluta 140 es 13).
- PC: Contenido del contador de programa
  `6, PC, 15` (En el instante 6 el contador de programa contiene el valor 15).
- ACC: Contenido del acumulador.
  `3, ACC, 10` (En el instante 3 el acumulador contiene el valor 10).
- IR_OP: Código de operación del registro de instrucción.
  `10, IR_OP, JUMP` (En el instante 10 el registro de instrucción contiene la operación JUMP_INST).
- IR_O1: Operando 1 del registro de instrucción.
  `10, IR_O1, 2` (En el instante 10 el operando 1 del registro de instrucción contiene el valor 2).
- IR_O2: Operando 2 del registro de instrucción.
  `10, IR_O2, 0` (En el instante 10 el operando 2 del registro de instrucción contiene el valor 0).
- PSW: Contenido del registro PSW.
  `16, PSW, 1` (En el instante 16 el registro PSW contiene el valor 1, en decimal).
- MAR: Contenido del registro MAR.
  `20, MAR, 140` (En el instante 20 el registro MAR contiene el valor 140).
- MBR_OP: Contenido del código de operación del registro MBR.
  `6, MBR_OP, ZJUMP` (En el instante 6 el código de operación del registro MBR tiene el valor ZJUMP_INST).
- MBR_O1: Contenido del operando 1 del registro MBR.
  `6, MBR_01, 150` (En el instante 6 el operador 1 del registro MBR 150).
- MBR_O2: Contenido del operando 2 del registro MBR.
  `6, MBR_02, 10` (En el instante 6 el operador 2 del registro MBR contiene el valor 10).
- MMU_BS: Contenido de registro base de la MMU.
  `6, MMU_BS, 60` (En el instante 6 el registro base de la MMU contiene el valor 60).
- MMU_LM: Contenido del registro límite de la MMU.
  `6, MMU_LM, 20` (En el instante 6 el registro límite de la MMU contiene el valor 20).
- MMU_MAR: Contenido del registro MAR de la MMU.
  `6, MMU_MAR, 7` (En el instante 6 el registro MAR de la MMU contiene el valor 7).
- MMEM_MAR: Contenido del registro MAR de la MainMemory.

`6, MMEM_MAR, 15` (En el instante 6 el registro MAR de la MainMemory contiene el valor 15).

- `MMBR_OP:` Contenido del código de operación del registro MBR de la MainMemory.

  `6, MMBR_OP, ADD` (En el instante 6 el código de operación del registro MBR de la MainMemory contiene el valor ADD_INST).

- `MMBR_O1:` Contenido del operando 1 del registro MBR de la MainMemory.

  `6, MMBR_O1, 3` (En el instante 6 el operando 1 de registro MBR de la MainMemory contiene el valor 3).

- `MMBR_O2:` Contenido del operando 2 del registro MBR de la MainMemory.

  `6, MMBR_O2, 5` (En el instante 6 el operando 1 de registro MBR de la MainMemory contiene el valor 5).

- `XPID:` Contenido del executingProcessID del OperatingSystem.

  `34, XPID, 0` (En el instante 34 el executingProcessID contiene el valor 0, en decimal).

- `RMEM`: Contenido almacenado en la posición de memoria relativa del proceso en ejecución indicada:
  `5, RMEM, 13, 3` (En el instante 5 el contenido que hay en la dirección relativa 3 es 13).

- `AMEM`: Contenido almacenado en la posición de memoria absoluta indicada:
  `5, AMEM, 127, 140` (En el instante 5 el contenido que hay en la dirección absoluta 140 es 127).

- `MBR:` Contenido del registro MBR de la MainMemory.

  `6, MBR, 46` (En el instante 6 el registro MBR de la MainMemory contiene el valor 46).

- `MMBR:` Contenido del registro MBR de la MainMemory.

  `6, MMBR, 53` (En el instante 6 el registro MBR de la MainMemory contiene el valor 53).

- `PCB_ST:` Contenido del campo status del PCB de índice indicado en address.

  `15, PCB_ST, 0, 1` (En el instante 15 el status del PCB de PID 1 contiene el valor 0).

- `PCB_PC:` Contenido del campo PC del PCB de índice indicado en address.

  `6, PCB_PC, 16, 0` (En el instante 6 el PC del PCB de PID 0 contiene el valor 16).

- `PCB_PR:` Contenido del campo priority del PCB de índice indicado en address.

  `6, MMBR, 100, 2` (En el instante 6 la priority del PCB de PID 2 contiene el valor 100).

Generación del fichero de asertos

El fichero de asertos puede realizarse a mano, como un fichero de texto normal conteniendo en cada línea un aserto según lo descrito, dejando constar en el fichero el valor que se estima que debería tener cada elemento a verificar, en los instantes temporales que se consideren.

Una manera más sencilla de generar el fichero de asertos es ejecutar el simulador con la opción de generar fichero de asertos `--generateAsserts`. Un fichero de asertos tiene que existir, con el mismo formato que antes, pero lo que hace en este caso es generar el aserto correspondiente para comprobar el valor del elemento indicado en el instante preciso. O sea, si en el fichero de asertos tenemos un aserto como

`6, PC, *cualquier valor*`

lo que se hará al ejecutar el simulador con la opción de generar asertos será generar un aserto similar al anterior:

`6, PC, *valor*`

pero donde el *valor* que se incluirá será el que realmente tenga el contador de programa en el instante 6. El "*cualquier valor*" que se incluía en el aserto original no se va a usar para nada, pero debe de ser del tipo adecuado dado que se utiliza la rutina normal para leer asertos del fichero.

Los asertos generados se muestran por pantalla (no se crea ningún fichero), por lo que tendrá que ser el usuario quien después copie y pegue lo que estime oportuno al fichero de asertos final.

Especificación de instantes temporales

En el fichero de asertos que se utilizará para generar los valores reales de los distintos elementos, puede utilizarse en cualquier aserto como instante temporal el carácter “*”: esto hará que ese aserto se genere para todos los instantes temporales. En realidad, este carácter también se puede utilizar al verificar los asertos, pero deberá hacerse sólo para elementos que no varíen con el tiempo (contenido de posiciones de memoria que no van a cambiar, por ejemplo).