Práctica 3: Ejecución de programas complejos

Fecha de entrega: 2 de Marzo, 16:00

Objetivo: Implementación y uso de la herencia, interfaces, excepciones y ficheros

1. Nueva funcionalidad

La TPMV en el estado dejado por la práctica 2 es bastante inmanejable para la ejecución de programas-mv que requieran la ejecución de más de 10-15 instrucciones. Además, los programas no permiten la interacción con el usuario, sino que son meros "computadores" de cálculos con aritmética entera.

El objetivo principal de esta práctica será permitir ejecutar programas-mv más complejos de forma transparente, es decir, de forma similar a la que se ejecutan programas en Java. Para eso haremos que la aplicación pueda cargar los programas a ejecutar desde ficheros externos (el código fuente en ensamblador) y éste se ejecute automáticamente sin necesidad de ir solicitando la ejecución individual de cada instrucción máquina.

Gracias a esta ejecución transparente, podemos extender la TPMV con una instrucción adicional para *leer* datos del teclado. En la práctica anterior esa instrucción no tenía sentido porque se "mezclaría" con los datos que se le pedían al usuario de la práctica (para el `STEP`, etc.), en contraposición con el usuario *del programa-mv*. Ahora, cuando la TPMV esté ejecutando un programa-mv de forma transparente y alcanza esa instrucción nueva (`IN`), la aplicación esperará a que el usuario escriba y lo "redirigirá" hacia el programa-mv. Como decimos, es similar a la ejecución de un programa en Java: cuando se lanza la JVM y ésta comienza a ejecutar el programa, cualquier cosa que el usuario teclee *no* es capturado e interpretado por la JVM, sino que esas pulsaciones son utilizadas por el propio programa Java.

De esta forma, podremos tener programas-mv que, por ejemplo, lean una lista de enteros, la ordenen y muestren por pantalla.

Por último, en la práctica 3 permitiremos al usuario la depuración completa del programa permitiendo la modificación del estado de la máquina virtual en medio de la ejecución

cuando se utiliza la ejecución interactiva. Por lo tanto, esta práctica requiere IMPLEMENTAR LAS PARTES OPTATIVAS DE LA PRACTICA 2.

A modo resumen, la lista de nuevas funcionalidades es la siguiente:

- Eliminaremos la necesidad de que el usuario tenga que teclear las instrucciones del programa-mv a ejecutar. En esta práctica leeremos el programa de un fichero. No obstante, se seguirá pudiendo escribir el programa como en la práctica anterior, directamente en la consola.

- Añadiremos unas pocas instrucciones nuevas a la TPMV. La más relevante es la instrucción `IN` que hace que la TPMV lea un carácter de la entrada del usuario. Otras dos están relacionadas con permitir el uso de vectores y la última posibilita la existencia de "procedimientos".

- Extenderemos la aplicación para tener dos modos de funcionamiento: el modo interactivo (el de la práctica anterior) donde el usuario va guiando el proceso de ejecución del programa-mv y el modo no interactivo (o transparente) en donde el programa-mv es ejecutado de principio a fin automáticamente.

- Permitiremos controlar cuál es la entrada y salida que utilizará el programa-mv. En particular, podremos indicar a la TPMV que la entrada y salida de datos debe hacerse con los métodos habituales (leer de teclado, escribir por pantalla) o utilizando ficheros.

- Mejoraremos los mensajes de error dados por la aplicación cuando alguna instrucción falla al ejecutar.

- Se implementan las partes optativas de la práctica 2.

El usuario podrá seleccionar el modo de funcionamiento concreto utilizando los parámetros de la aplicación. Las posibilidades son las siguientes (se muestran cuando se utiliza el parámetro `-h` o `-help`):

```
$ java -cp [detalles omitidos] tp.mv.Main -h

usage: tp.pr3.mv.Main [-a <asmfile>] [-h] [-i <infile>] [-m <mode>] [-o
       <outfile>]
 -a,--asm <asmfile>   Fichero con el codigo en ASM del programa a
                      ejecutar. Obligatorio en modo batch.
 -h,--help            Muestra esta ayuda
 -i,--in <infile>     Entrada del programa de la maquina-p.
 -m,--mode <mode>     Modo de funcionamiento (batch | interactive). Por
                      defecto, batch.
 -o,--out <outfile>   Fichero donde se guarda la salida del programa de la
                      maquina-p.
```

Si se intenta utilizar la aplicación con parámetros incorrectos (no se indica fichero ASM, se utiliza una opción no válida, el fichero de entrada no existe, etc.), se mostrará un mensaje de error (en la salida de error, `System.err`) y se terminará la aplicación con código de error 1 (`System.exit(1)`).

Si el usuario especifica correctamente todos las opciones pero hay algún error al leer el fichero ASM, se terminará con código de error 2.

2. Nuevas instrucciones

En esta práctica añadiremos cuatro instrucciones nuevas:

- Instrucción de entrada de usuario, `IN`: es la operación *simétrica* a `OUT`. Lo que hace es leer un carácter de la entrada y apilarlo (convertido a entero) en la pila. Es una operación, pues, que no tiene parámetros en ensamblador ni necesita que haya operandos en la pila. Si se ha llegado al *final de la entrada*, se apila un -1. La operación podría fallar si hay algún error en la lectura (típicamente, cuando se están simulando las pulsaciones de teclas leyendo de un fichero y se produce algún error al leer de él).

- Instrucción de salto indirecto, `JUMPIND`: todas las operaciones de salto que tenemos hasta ahora son *estáticas*, en el sentido de que la dirección de salto está prefijada en el código. Esta nueva instrucción permite decidir el lugar al que se salta en el momento de la ejecución. Para eso, `JUMPIND` *lee de la cima de la pila* el valor del nuevo contador de programa en lugar de cogerlo desde el parámetro de la propia instrucción. Antes de saltar elimina el dato de la pila.

- Lectura indirecta de memoria, `LOADIND`: de forma similar a la anterior, la única forma de leer de memoria que tenía la `TPMV` en la práctica 2 era fijando la dirección de lectura en el código ensamblador. Esta nueva operación permite calcular la dirección durante la ejecución, de forma que se lee de la posición de memoria indicada en la cima de la pila.

- Escritura indirecta en memoria, `STOREIND`: la operación simétrica a la anterior y relacionada con `STORE`, escribe en la posición de memoria indicada por la subcima el valor que aparece en la cima de la pila.

3. Formato del fichero con el programa-mv

El programa-mv que la `TPMV` carga (indicado con el parámetro `-a`) es un fichero de texto donde cada línea contiene una instrucción con el mismo formato que el utilizado en las prácticas anteriores.

Existe únicamente una capacidad adicional: el programa puede contener comentarios. El principio de los comentarios es marcado con el carácter '`;`' (que deberá aparecer como primer carácter en la línea o ir precedido siempre de un espacio), y debe interpretarse como que a partir de él la línea completa debe ignorarse.

Un ejemplo de programa válido aparece a continuación. Gracias a los comentarios, es fácil entender qué es lo que hace:

```
; Mini programa que lee toda la entrada y la vuelve
; a escribir eliminando todas las aes minúsculas.
; En C++ sería algo así:
;
; while (cin) {
;     char c;
;     cin << c;
;     if (cin && (c != 'a'))
;        cout >> c;
; }
```

```
;

; Leemos el caracter de la entrada
in
; Miramos si hemos llegado al final
dup      ; Duplicamos el valor devuelto por IN
push -1  ; Si es == -1, entonces
eq       ; hemos llegado al final de la entrada
bt 11    ; por lo que saltamos al final del programa
; No hemos llegado al final. Miramos si es una 'a'
dup
push 97  ; ascii('a') = 97
eq
bt 0
; No es una 'a'. Escribimos y volvemos a empezar
out
jump 0
;
; Fin del programa.
; Número de instrucciones = 11
;
```

4. Entrada y salida del programa-mv

Como ya hemos dicho, las instrucciones `IN` y `OUT` de la `TPMV` permiten que ésta ejecute programas que piden al usuario datos (leen caracteres de entrada) o escriben información (escriben en la salida).

En el caso del modo interactivo, pueden surgir dudas. Por ejemplo, cuando el usuario teclea un comando por consola, ¿ese comando es parte de la entrada del programa? ¿Y cuando se imprime por pantalla el estado de la MV, es este mensaje parte de la salida? Por establecer un criterio:

- En el modo interactivo y batch es parte de la entrada del programa-mv exclusivamente lo que contenga el fichero de entrada indicado al main con el argumento -i. En modo batch, si no se indica el argumento -i, la entrada estándar coincide con la que se suministra a la aplicación.

- En el modo interactivo y batch, es parte de la salida del programa-mv aquello que genere como salida la instrucción OUT. Esta salida se volcará en el fichero especificado el argumento -o. En el caso de la ejecución no interactiva (batch), si no se indica la opción -o se utilizará la salida estándar.

- Los errores de ejecución (o errores "hardware") provocados por programas-mv incorrectos o cualquier otra causa se escribirán por la salida de error (`System.err`).

- Cuando se ejecute la aplicación en modo interactivo, si no se indica fichero de entrada (parámetro `-i`), el programa-mv *no* tendrá entrada, por lo que las instrucciones `IN` apilarán siempre un -1. De forma similar, si no se especifica fichero de salida (parámetro `-o`), no habrá salida, por lo que la instrucción `OUT` no tendrá efecto.

En la sección anterior apareció un programa que lee de la entrada y va escribiendo lo que lee, con excepción del caracter 'a'(código ASCII 97). Si ejecutamos ese programa-mv

y tecleamos lo siguiente[1]:

```
Esta es la entrada de ejemplo.
El programa no escribira las aes.
```

el programa-mv escribirá

```
Est es l entrd de ejemplo.
El progrm no escribir ls es.
```

Es decir, escribirá el texto sin los caracteres 'a'.

Como ya se describió en la sección 1, podemos modificar el comportamiento de la TPMV para que en lugar de esperar que el usuario teclee el texto, *lo lea* de un fichero.

Así, si el código del programa-mv está en el fichero `quitaAes.asm` y grabamos el texto anterior en el fichero `entrada.txt`, podemos ejecutar la práctica con:

```
$ java -cp [detalles omitidos] tp.mv.Main -a quitaAes.asm -i entrada.txt
```

Y el resultado será el mismo que si lo hubiéramos introducido de forma manual.

Como ya hemos explicado antes, el funcionamiento de los parámetros `-i` y `-o` depende en realidad del modo de ejecución utilizado. En concreto:

- Si se está utilizando el modo no interactivo (modo `batch`, utilizado por defecto) y *no* se especifica el parámetro `-i` o el parámetro `-o` la TPMV utilizará la entrada estandar y salida estandar respectivamente como entrada y salida del programa-mv.
- Si se está utilizando el modo interactivo y no se especifica los parámetros `-i` o `-o`, *no existirá* entrada ni salida del programa-mv, es decir, si el programa-mv intenta leer de la entrada, la instrucción `IN` devolverá -1 (`EOF`) y la instrucción `OUT` no tendrá efecto. La razón es fácil de entender: al ejecutarse en modo interactivo, la aplicación está mostrando información al usuario y preguntándole por las acciones a realizar (STEP, RUN, ...). Por lo tanto la entrada y salida del programa-mv se mezclaría con las de la propia aplicación interactiva.

5. Errores "hardware"

Cuando la TPMV intenta ejecutar alguna instrucción y ésta falla, se produce una excepción en el "hardware" pues se le ha pedido que ejecute algo imposible. En las prácticas anteriores esos errores (que, usando la nomenclatura típica usada en el *hardware*, se llaman *traps*) se mostraban al usuario con un poco significativo "Error al ejecutar la instrucción".

En esta práctica cuando ocurra un error de este tipo, el mensaje debe dar una explicación personalizada al fallo producido[2]:

- Cuando la instrucción necesita elementos en la pila que no están. Se mostrará un mensaje del tipo "Error ejecutando ADD: faltan operandos en la pila (hay 1)".
- Cuando se intenta realizar una división por cero se mostrará un simple "División por cero".

[1] Para que el programa-mv termine, tendrás que *cerrar la entrada*, para que la instrucción `IN` apile el -1, algo que no es posible en algunos terminales. En los terminales de Linux puede hacerse utilizando Ctrl-D.

[2] Ver ejemplo de ejecución concretos en la sección 6.

- Cuando falla una operación de lectura o escritura por problemas inherentes a la propia entrada o salida (se está leyendo de un fichero y la lectura ha fallado, o se está escribiendo la salida a disco y no hay espacio, por ejemplo), se mostrará un mensaje como "Error ejecutando OUT:", seguido de la causa del error (mensaje contenido en la `IOException` subyacente).

En todos los casos ese mensaje de error se escribirá por la salida de error (`System.err`). Si se produce uno de estos errores cuando se está ejecutando en modo no interactivo (`batch`), la ejecución terminará con un error (código de error 2).

6. Ejemplos de ejecución

6.1. Ejemplo de uso incorrecto de parámetros

La aplicación debe validar que los parámetros son utilizados de forma correcta. Por ejemplo:

- En el modo no interactivo o batch (modo por defecto), se debe especificar el fichero ensamblador a ejecutar[3]:

```
$ java -cp [detalles omitidos] tp.mv.Main

Uso incorrecto: Fichero ASM no especificado.
Use -h|--help para más detalles.
```

- Se debe garantizar que tras la opción `-a`, `-o` y `-i` viene el nombre de un fichero:

```
$ java -cp [detalles omitidos] tp.mv.Main -o -a quitaAes.asm

Uso incorrecto: Missing argument for option: o
Use -h|--help para más detalles.
```

- Se debe mostrar un error personalizado cuando se intenta cargar un fichero en ensamblador incorrecto. Por ejemplo, si el fichero es:

```
push 78
OUT
apila 29
out
```

podría ser algo como:

```
$ java -cp [detalles omitidos] tp.mv.Main --asm incorrecto.asm

Error en el programa. Linea: apila 29
```

- También se dará algún mensaje de error si se especifica un fichero de entrada que no existe[4]:

[3] Los mensajes que se escriben en la salida de error, `System.err`, aparecen en rojo y cursiva para distinguirlos del resto.

[4] No ocurrirá así si es un fichero de salida, pues el fichero se creará nuevo.

```
$ java -cp [detalles omitidos] tp.mv.Main -a quitaAes.asm --in noExiste.asm

Uso incorrecto: Error al acceder al fichero de entrada (noExiste.txt)
Use -h|--help para más detalles.
```

- Como último ejemplo, debemos comprobar que el modo de ejecución que se selecciona es uno de los permitidos:

```
$ java -cp [detalles omitidos] tp.mv.Main -a quitaAes.asm -m invalido

Uso incorrecto: Modo incorrecto (parametro -m|--mode)
Use -h|--help para más detalles.
```

6.2. Ejemplos de errores hardware

Cuando las instrucciones fallan al ejecutarse, se debe mostrará un mensaje de error. El siguiente ejemplo muestra una traza de ejecución en modo interactivo sin uso de fichero ASM, sino donde el usuario primero introduce el programa (incorrecto) y después lo ejecuta paso por paso. Se puede ver la respuesta de la aplicación al ejecutar la instrucción ADD cuando falla por falta de operandos:

```
$ java -cp [detalles omitidos] tp.mv.Main -m interactive

Introduce el programa fuente
> push 8
> add
> end
El programa introducido es:
0: PUSH 8
1: ADD
> STEP
Comienza la ejecución de PUSH 8
El estado de la máquina tras ejecutar la instrucción es:
Memoria: <vacía>
Pila de operandos: 8
> STEP
Comienza la ejecución de ADD
Error ejecutando ADD: faltan operandos en la pila (hay 1)
> PUSH 9
> STEP
Comienza la ejecución de ADD
El estado de la máquina tras ejecutar la instrucción es:
Memoria: <vacía>
Pila de operandos: 17
```

Un error parecido ocurre cuando se ejecuta una instrucción de lectura de memoria sobre una dirección incorrecta. El programa siguiente escribe una 'A' antes de fallar:

```
PUSH 65
OUT
LOAD -2
```

Si se ejecuta en modo no interactivo (batch), tras mostrar la 'A', aparece el mensaje de error personalizado:

```
A***Error ejecutando LOAD -2: dirección incorrecta (-2)***
```

Como último ejemplo de error "de hardware", ejecutamos el siguiente programa (que generará una división por 0):

```
PUSH 10
PUSH 0
DIV
```

Cuando la ejecución la realizamos de forma interactiva tenemos como resultado:

```
> run
Comienza la ejecución de PUSH 10
El estado de la máquina tras ejecutar la instrucción es:
Memoria: <vacía>
Pila de operandos: 10
Comienza la ejecución de PUSH 0
El estado de la máquina tras ejecutar la instrucción es:
Memoria: <vacía>
Pila de operandos: 10 0
Comienza la ejecución de DIV
División por cero
> quit
```

7. Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha y hora indicada en la cabecera de la práctica.

El fichero debe tener al menos el siguiente contenido[5]:

- Directorio `src` con el código de todas las clases de la práctica.
- Fichero `alumnos.txt` donde se indicará el nombre de los componentes del grupo.

[5] Puedes incluir también opcionalmente los ficheros de información del proyecto de Eclipse