Taller de Programación I (75.42)
Facultad de Ingeniería
Universidad de Buenos Aires

Remote Java Virtual Machine

Ejercicio N° 1

Objetivos	• Buenas prácticas en programación de Tipos de Datos Abstractos (TDAs) • Modularización de sistemas • Correcto uso de recursos (memoria dinámica y archivos) • Encapsulación y manejo de Sockets
Instancias de Entrega	**Entrega 1:** clase 4 (11/09/2018). **Entrega 2:** clase 6 (25/09/2018).
Temas de Repaso	• Uso de structs y typedef • Uso de macros y archivos de cabecera • Funciones para el manejo de Strings en C • Funciones para el manejo de Sockets
Criterios de Evaluación	• Criterios de ejercicios anteriores • Cumplimiento de la totalidad del enunciado del ejercicio • Ausencia de variables globales • Ausencia de funciones globales salvo los puntos de entrada al sistema (*main*) • Correcta encapsulación en TDAs y separación en archivos • Uso de interfaces para acceder a datos contenidos en TDAs • Empleo de memoria dinámica de forma ordenada y moderada • Acceso a información de archivos de forma ordenada y moderada

Índice

Introducción

En el presente trabajo se implementará una máquina virtual de Java reducida y remota.

Descripción

La máquina virtual de Java [1] (o JVM por sus siglas en inglés) remota consta de un único ejecutable con dos modos de operación: **`client`** y **`server`** (cliente y servidor).

El cliente leerá una serie de instrucciones conocidas como byte codes y se las enviará al servidor para que el servidor ejecute las ejecute junto con el tamaño del *array de variables* **`N`**.

En particular, el cliente se conectara al servidor usando TCP, le enviará primero el tamaño del *array de variables* **`N`** en un ***entero con signo de 4 bytes big endian***, luego le enviara los byte codes y finalmente cerrara la conexion parcialmente, cerrando ***el canal de escritura*** del socket.

En esta versión reducida de Java, la máquina virtual implementada en el servidor mantendrá en memoria un *array de variables* y un *stack de operandos*.

El *array de variables* tiene una dimensión fija de **`N`** números enteros **`int`** inicializados a **`0`**, donde **`N`** es un parámetro del programa *cliente* y es recibido por socket por el *servidor*.

El *stack de operandos* inicia junto con el programa Java vacío e irá creciendo o achicandose de forma dinámica según se requiera.
De él se tomarán los argumentos para las distintas operaciones como así se lo usará para guardar los resultados. Todos los elementos del *stack* serán enteros **`int`**.
Al finalizar el programa todo el *stack* es eliminado perdiéndose todos los valores que en él están almacenados.

A medida que el servidor va ejecutando cada instrucción esta debe ser impresa por ***salida estándar*** por su nombre simbólico.

Al finalizar la ejecución el servidor además deberá imprimir por ***salida estándar*** el contenido del *array de variables* imprimiendo cada número en su propia línea en notación ***hexadecimal de 8 dígitos***.

> Nota: leer la documentación de **`printf`** [2]

Luego, le enviará al cliente el contenido del *array de variables* enviando cada número como ***4 bytes con signo en big endian***. El servidor no necesita enviarle al cliente cuántos elementos tiene el *array* ya que el cliente eso lo sabe.

El cliente recibe el *array* y lo imprime por ***salida estándar*** de igual manera que lo hizo el cliente.
Luego, ambos finalizan la conexión TCP y finalizan su ejecución: el servidor solo procesa un único cliente.

Instrucciones de la máquina virtual

Las instrucciones que la máquina de Java [3] debe soportar son las siguientes. Se indican cuantos elementos del *stack de operandos* se sacan o se ponen, cuantos *byte codes* adicionales se leen y que operación realizan.

`iand`, **`ior`**, **`ixor`**: retira dos elementos del *stack de operandos*, realiza la operación bit a bit (*and*, *or*, *xor*) y guarda el resultado en el mismo *stack*.

`iadd`, **`isub`**, **`imul`**, **`idiv`**, **`irem`**: retira dos elementos del *stack del operando*, realiza la operación aritmética (*suma*, *resta*, *multiplicación*, *división*, *módulo* o *resto*) y guarda el resultado en el mismo *stack*.
La operación es siempre entre enteros (**`int`**) y el resultado es siempre un **`int`**. En el caso de la división y del módulo, se garantiza que el divisor siempre será distinto de 0.

`dup`: duplica el último elemento del *stack de operando* sin sacarlo y guarda la copia en el mismo *stack*.

`bipush`: lee el siguiente *byte code*, lo extentiende a un **`int`** con signo y lo guarda en el *stack de operandos*.

`istore`, **`iload`**: lee el siguiente *byte code*, y lo interpreta como un número sin signo. Este será el índice para seleccionar una variable del *array de variables* del programa Java (se garantiza que los índices son válidos). **`istore`** retira un elemento del *stack de operandos* y lo guarda en la variable indexada, **`iload`**, por el contrario, lee la variable indexada y la guarda en el *stack de operandos*.

Byte codes

A continuación se definen los byte codes que la máquina de Java deberá reconocer. Se muestran los nombres simbólicos a la izquierda y el número o código a la derecha, en hexadecimal.

```
istore 0x36
iload  0x15
bipush 0x10
dup    0x59
iand   0x7e
ixor   0x82
ior    0x80
irem   0x70
ineg   0x74
idiv   0x6c
iadd   0x60
imul   0x68
isub   0x64
```

Formato de Línea de Comandos

El cliente tiene la siguiente línea de comandos:

```
./tp client <host> <port> <N> [<filename>]
```

Donde **<host>** y **<port>** son la dirección IPv4 o *hostname* y el puerto o servicio donde el servidor estará escuchando la conexión TCP.

<N> es el tamaño del *array de variables*.

<filename> es un argumento opcional que indica el ***archivo binario*** con los *byte codes*. Si el argumento no es pasado, el cliente leerá los *byte codes* de la ***entrada estándar***.

El servidor tiene la siguiente línea de comandos:

```
./tp server <port>
```

Dónde **<port>** es el puerto o servicio donde el servidor estará escuchando la conexión TCP.

Códigos de Retorno

Tanto el cliente como el servidor deben retornar **0** si todo salió correctamente o **1** en caso contrario.

Entrada y Salida Estándar

El cliente leerá por ***entrada estándar*** los *byte codes* a ejecutar a menos que recibe el parámetro opcional (véase **Formato de Línea de Comandos**)

Por ***salida estándar*** imprimirá el *array de variables* luego de la ejecución del programa Java.

El servidor no leerá nada por ***entrada estándar*** e imprimirá las instrucciones a medida que las ejecuta. Luego imprimira el *array de variables* al igual que lo hará el cliente.

Véase **Ejemplo de Ejecución**.

Ejemplo de Ejecución

El cliente se conecta al servidor que está corriendo en **localhost**, puerto **8080**, y le envía **4** como el tamaño del *array de variables*. Luego lee por ***entrada estándar*** los *byte codes* y se los envía también.

```
echo -e '\x10\x41\x59\x59\x36\x00\x10\x01\x60\x59\x59\x36\x01' | ./tp client localhost 8080 4
```

Y su salida por salida estándar es:

```
Variables dump
00000041
00000042
00000000
00000000
```

El servidor se ejecuta como:

```
./tp server 8080
```

Y su salida por salida estándar es:

```
Bytecode trace
bipush
dup
dup
istore
bipush
iadd
dup
dup
istore

Variables dump
00000041
00000042
00000000
00000000
```

Ejecución paso a paso del código Java

A continuacion se detalla paso a paso la interpretacion del codigo Java y su ejecución.

Inicialmente el programa Java tendrá, en este ejemplo, **`4 ints`** en el *array de variables* inicializados a 0 y un *stack de operandos* vacío.

`\x10\x41` **`bipush:`** se agrega al stack el entero `0x41` (**`int`** con signo)
`\x59` **`dup:`** copia el último elemento del stack. Ahora el stack es | `0x41` | `0x41` |
`\x59` **`dup:`** ahora el stack tiene tres elementos: | `0x41` | `0x41` | `0x41` |
`\x36\x00` **`istore:`** saca el último elemento del stack y lo guarda en la variable **`0`**.

El *array de variables* en este momento es:

| 0x41 | 0x00 | 0x00 | 0x00 |

Y el *stack de operandos* es

| 0x41 | 0x41 |

`\x10\x01` **`bipush:`** agrega el número `0x01`. Stack actual: | `0x41` | `0x41` | `0x01` |
`\x60` **`iadd:`** toma los últimos 2 elementos del stack y los reemplaza por su suma.
Stack de operandos:
| `0x41` | `0x42` |

`\x59` **`dup:`** Stack: | `0x41` | `0x42` | `0x42` |
`\x59` **`dup:`** Stack: | `0x41` | `0x42` | `0x42` | `0x42` |
`\x36\x01` **`istore:`** toma el último elemento del stack y lo guarda en la variable **`1`**.

El *array de variables* final es:
| `0x41` | `0x42` | `0x00` | `0x00` |

Y el *stack de operandos* es
| `0x41` | `0x42` | `0x42` |

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C (C99).
2. Está prohibido el uso de variables globales.
3. El protocolo debe respetarse incluyendo pero no limitado a el formato de los mensajes, los tamaños, el endianess, el signo y el orden.

Referencias

[1] https://en.wikipedia.org/wiki/Java_virtual_machine
[2] http://www.cplusplus.com/reference/cstdio/printf/
[3] https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-6.html