INF 212 Analysis of Programming Languages
Project 8 – Virtual Machines
Due date: 6/3
Demo date: 6/4

Goal: Get acquainted with a virtual machine. Similarly to the type checking homework, the concrete task in this homework is very simple, but it requires you to analyze and understand an existing Java VM implementation. The task is just a means to the end goal, and not the goal in itself.

In process of achieving the goal, students with Windows machines will get the extra benefit of getting acquainted with another kind of virtual machine: an entire operating system virtual machine. For this one, you will not be looking inside, you will only install and use it. Students with Macs and Linux will not need this (and hence miss the opportunity to use it...).

Description

You will be using JamVM, an open source Java virtual machine (http://jamvm.sourceforge.net/). Start by downloading the latest release.

In this project you will create new Opcodes for JamVM. Opcodes 248-255 are unused by JamVM and thus are the Opcodes that you will be adding additional functionality for. A testcase is provided for you (here). Since the compiler will not be aware of your new opcodes, you will need to edit your .class files manually using a binary/hex editor if you wish to create different test cases.

The new opcodes will replace the IADD opcode during a standard arithmetic expression. For an expression

x + y

you need to replace the byte representing IADD (60 in hexadecimal) with the value of the new opcode. Therefore you can make the same assumptions about the state of the stack that you would for an integer addition expression. The modified bytecode would be analogous to

x OP y

if the compiler were to recognize the operator.

Setup

JamVM is only supported by Linux/Mac OS X/BSD/Solaris, not Windows. Windows users without access to a linux machine can install VirtualBox (http://www.virtualbox.org/) for free and run whatever distro they prefer in a (different kind of) virtual machine. Ubuntu 32 bit edition (http://www.ubuntu.com/download/ubuntu/download) is recommended. Students having any problems installing either GNU ClassPath or JamVM are recommended to try in a clean virtual Ubuntu installation. This project was tested and created in such an environment.

You will first need to download and install the GNU Classpath 0.98 from http://www.gnu.org/software/classpath/. It is recommended you compile this with the following configurations.

./configure --disable-gtk-peer --disable-gconf-peer --disable-plugin --disable-examples --disable-tools --enable-jni --disable-Werror

Then download and install JamVM from http://jamvm.sourceforge.net/. To make things simpler, disable caching in JamVM with the following configuration.

./configure--disable-int-caching

IMPORTANT: Verify that caching is turned off after installation by typing "jamvm –version." Under Execution Engine, it should say "inline-threaded interpreter" and not mention caching.

TASK: New Opcodes

248: May I Have Another?

x OP y -> (x + y) + 1

Example: 4 OP 11 = 16

249: Elite Ops

x OP y -> 1337

Always returns the same value.

250: Yes, I'm Sure I Want To Do That

x OP y -> x / y

Same as divide but division by zero will not throw exception. x / 0 will be treated as x / 1.

251: Reverse Div

x OP y -> y / x

You MUST check for division by zero

Example: 5 OP 50 = 10

252: Sum of Squares

x OP y -> x^2 + y^2

Examples: 10 OP 3 = 109

253: Stay a WHILE and listen

x OP y -> Fibonacci(x) + Factorial(y)

Example: 7 OP 5 = 133

254: Crash and Burn

Accessing the virtual machine at such a low level is very volatile, as you may have already found out. As your final test, you must crash your program when this opcode is read. It doesn't matter what kind of crash (exception, segmentation fault), but your JVM must NOT return a result and must halt.

Tips:

You must acquire a general idea of how the VM is implemented. The task involves edits to jam.h, interp/direct.c, interp/shared.h, interp/engine/interp.c, interp/engine/interp-threading.h

Demo

The discussion during the demo will revolve around your understanding of the JamVM, opcodes and beyond.