lecture 25

Java Virtual Machine
(different from MIPS virtual machine)

Wed. April 13, 2016

- what is the JVM?
- what is Java "byte code"
(is it related to MIPS assembly/machine code?)
- what are .class files?
- what is garbage collection?

"Virtual Machine" (MIPS, Java) can refer to:
1) an abstract specification
   - a well defined instruction set
     (both assembly and machine code)
   - specified "on paper"
2) an implementation
   - software that runs on your computer (e.g. MARS for MIPS)
3) a runtime instance of that software

Example: what you've seen in COMP 273

Let's turn our attention specifically to the Java Virtual Machine (JVM).

"Java API" can refer both to:
1) an abstract specification
   - set of classes and interfaces (with methods & fields)
   - on paper
2) an implementation (libraries)
   i.e. class files (see previous slide)
Run time

your compiled classes
Java API classes

The above classes are “loaded” into the JVM at runtime, as they are needed.

Java Virtual Machine

implementation (assembly language or machine code for your processor i.e. "native" code)

Java History (early 1990s)
- Java language created (by James Gosling at Sun)
- first web browsers written (html created) (*Java applets run in browser*)

Why was Java so important for www?
- portability
  (downloaded .class files can run on any JVM)
- security
  (your compiled Java code doesn’t know which computer it is running on. The platform provides a layer of protection (unlike C code which uses memory addresses explicitly)

Let’s compare MIPS with Java (and then JVM)

MIPS
data
- registers
- Memory
  - stack (0xffff ffff)
  - globals & heap (0x1000 0000)

instructions
- Memory
  - text

Java (high level)
classes
- fields (data)
- methods (instructions)
- superclass

objects (instances of classes)

JVM

Everytime a method is invoked, a stack frame is added.
Stack frames correspond to methods, like in MIPS. Stack frames contain:
- a pointer to a method
- a program counter within that method. (This is quite different from what you are used to in MIPS.)
- local variables of the method, e.g. references to objects.
- an operand stack (a stack within a stack) which I will explain later.

"methods" point to information about the class to which it belongs (not shown)

Details are very implementation dependent. Think of the above as data structures in a JVM, namely in assembler code of the real computer.

Portability

your Java programs
Java platform
- Linux
- Mac OS
- Windows
Processor X
Processor Y
Processor Z

The Java platform and the OS both can be written in a higher level language (C, C++) but ultimately they must be compiled down to assembly language or machine code for particular processor.

Why was Java so important for www?
- portability
  (downloaded .class files can run on any JVM)
- security
  (your compiled Java code doesn’t know which computer it is running on. The platform provides a layer of protection (unlike C code which uses memory addresses explicitly)

MIPS (assembly)

data
- registers
- Memory
  - stack (0xffff ffff)
  - globals & heap (0x1000 0000)

instructions
- Memory
  - text
MIPS instructions

JVM instructions
("byte code")

MIPS
registers and Memory

JVM
No registers (except PC)

Indeed, no CPU.

Uses "operand stack" (stack within a stack)

Example 1

How is this done?
MIPS analogy: load words from Memory into registers, then perform operation, then write back to Memory.

Java stack frame (over time)

local variable index in the method's stack frame

Example 2

class SumToN {
    public static int sum(int n){
        int sum = 0;
        for (int k=0; k < n; k++){
            sum = sum + k;
        }
        return sum;
    }
}

What is in a class file? (on disk)
- 4 bytes 0xCAFEBABE
- constants e.g. numbers, strings, identifier (names)
- fields (including types)
- methods (byte code)

What defines a class?
- name
- fields (identifier, type)
- methods (local variables, instructions, return type)
- modifiers

Try it yourself (linux)
javac SumToN.java
produces SumToN.class
javap -c SumToN
"disassembles" the class file
number of methods: 67
number of bytes in method 0: 4
byte code for method 0: 52
number of bytes in method 1: 94
byte code for method 1: 45
number of bytes in method 2: 45
byte code for method 2: 67
number of bytes in method 3: 94
byte code for method 3: 4

Garbage Collection

As more and more objects are created, the heap eventually "fills up". What to do?

(Depends on implementation. The stack and heap should not be thought of as part of a single virtual address space, as we have seen in MIPS.)

Define a graph

Garbage Collection: "Mark and Sweep" Algorithm

- mark each object vertex as not visited
- for each reference variable (vertex) in the stack
  - traverse the graph starting from that vertex, and mark each object that you visit as visited
- remove each not visited object vertex (garbage)

JVM maintains a linked list of objects. Only live (non-garbage) need to be kept.

BEFORE GC

AFTER GC

If two objects point to each other, but nothing points to either of them, then both will be removed by garbage collection.

Final Exam

48 multiple choice questions (5 choices per question):
- 18 have number / bitstring / hex / boolean formula answers
- 6 are of the form 'which of these is not correct?'
- the rest require you to choose the best (positive) answer e.g. which of the following is correct?

Answer them all. If you are unsure, then eliminate as many as you can and guess from the rest.

This approach is consistent with principle of "no negative marking":

The issue is subtle. Some multiple choice exams penalize you for getting wrong answer, but they don’t penalize you for leaving a question blank. This encourages you to do probabilistic calculations on the exam, based on your certainty of whether you are correct or not. This is a distraction (not good).

Instead of penalizing you for guessing, I will scale the grades downward, so you need to answer correctly more than half the questions correctly in order to get a grade of 50%. To my knowledge, such scaling is not controversial.