VU Abstrakte Maschinen
Java Virtual Machine Prototype

Thomas Ogrisegg

Fakultät für Informatik
Technische Universität Wien

25. Juni 2015

Goals

- Implement a JVM with qemu that allows to:
- 1) run Java program that writes *Hello, World* to stdout
- 2) run Java program that does some more complex integer operations
- 3) compare performance with other JVMs

However...

- getting qemu to work with non-ELF binary formats is a bit more complex...
- Underestimated reading java specs...
- Only an interpreter was implemented (no JIT-compiler)

Implementation / TODO

- x class loader
- x jar-Loader
- - implemented instructions: control and integer only
- o Class/Object instantiation and inheritance
- o native functions
- o exceptions
- o monitors (synchronization) / multi-threading

Hello, World

```
public class hello {
   public static native void printOut (String s);

   public static void main (String args[]) {
      printOut ("Hello, World");
   }
}
```

Performance Test (PRNG)

```
public static int runme(int i)
{
    int ints[] = new int[i];
    int res = 0;
    int r = 23987837, z = 4829141;
    ints[0] = 23;

    /* generate random numbers */
    for (int x=1;x<i;x++) {
      r = r * z * x;
      ints[x] = r;
      z ^= r;
      z *= 53;
      r += 42;
    }
    qsort (ints, 0, ints.length-1);
    for (int x=0;x<i-1;x++) {
      if (ints[x] > ints[x+1]) {
        System.err.println ("Error!");
      }
    }
    return ints[ints.length/2];
}
```

Performance Test (qsort)

```
public static void qsort(int ints[], int y, int j)
{
    int low = y, high = j;
    int pivot = ints[(low + (high - low)/2)];
    while (y <= j) {
      while (ints[y] < pivot)
         y++;
      while (ints[j] > pivot)
         j--;
      if (y <= j) {
         int tmp = ints[y];
         ints[y] = ints[j];
         ints[j] = tmp;
         y++;
         j--;
      }
    }
    if (low < j)
      qsort (ints, low, j);
    if (y < high)
      qsort (ints, y, high);
}
```

Performance Test

- for each i in (1 10 100 1000 10000 100000 1000000 10000000) do 32 runs of
- 1) Oracle Java JIT-JVM
- 2) this Java interpreter
- measure min, max, avg of each run
- gives feeling of scale of JIT vs. interpretation

Tabelle: Performance

i	Interpreter			Oracle JVM		
	min	max	avg	min	max	avg
1	0.0010	0.0020	0.0010	0.0500	0.0660	0.0574
10	0.0010	0.0010	0.0010	0.0510	0.0750	0.0583
100	0.0020	0.0020	0.0020	0.0490	0.0720	0.0588
1000	0.0080	0.0110	0.0081	0.0500	0.0780	0.0610
10000	0.0380	0.0500	0.0437	0.0650	0.0840	0.0744
100000	0.4260	0.4400	0.4326	0.0680	0.1100	0.0947
1000000	4.9160	5.0610	4.9464	0.1680	0.1940	0.1794
10000000	54.2040	54.9170	54.4600	1.2310	1.2810	1.2542