Sequence 7.1 – Virtual machines

P. de Oliveira Castro S. Tardieu

Universal executables

- Software editors would love to write software once and execute it everywhere.
- The variety of processor families (Intel, ARM) and operating systems (GNU/Linux, Windows, OSX, Android, iPhone) make this difficult.
- One could write code for only one processor and operating system, and users would use emulators to emulate this combination.
- Unfortunately, full operating systems have not been designed with easy and efficient emulation in mind.

Universal platform

- What if we could design a platform with all useful services (graphical user interface, file operations, network programming, etc.)?
- This platform would be well-defined and simple enough so that an emulator could be easily written on any common operating system.
- We could compile our code for this virtual platform and execute it through the emulator on the user's computer.
- Such platforms already exist: the Java virtual machine (JVM) and .NET (by Microsoft).

The Java virtual machine

- Released in 1996 by Sun Microsystems (now Oracle).
- Slogan: write once, run anywhere.
- Both a new programming language (Java) and an execution platform (JVM) with rich libraries (GUI, network, etc.).
- Use instructions called *bytecodes*.
- Open specifications of the JVM to ease implementation by third-party vendors.
- JVM integration in web browsers to run *applets*.

Problems appeared soon after:

- Different release cycles between browsers and Java versions $\Rightarrow$ solved by creating compiler *plug-ins* upgradable independently.
- Very poor performances compared to native compiled languages $\Rightarrow$ on-site optimization.

On-site optimization

- Emulating a virtual machine is slow compared to a native program.
- Since Java programs are compiled into bytecodes that are then interpreted by the JVM, the bytecodes can themselves be seen as a well specified programming language (although in binary form rather than in text form).
- It is possible to compile the bytecodes into native code for the user computer and attain native compiled code performance.
- Calls to JVM libraries (GUI, network, etc.) can be replaced by calls to the user operating system libraries.

The program is still written and distributed for one platform (the JVM), and optimized specifically for the user machine.

Local recompilation

- Good compilation is slow because of heavy computations due to optimization passes.
- It is possible to detect frequently called functions at run-time and compile them on-the-fly (*just-in-time compilation*, or *JIT*). The infrequently called code will not be compiled.
- It is possible to compile the bytecode at installation time. This is what Android does when a new application is installed.
- Since the target processor is known when compiling the bytecode, it is possible to generate very precisely targeted code (not all Intel processors have the same vector processing instructions and registers for example).

So everyone must write in Java?

- The JVM has been designed with the Java programming language in mind.
- However, the JVM never sees any Java code, only bytecodes.
- If a compiler produces bytecodes, its output can be executed on the JVM.
- Several compilers target the JVM: GNAT (Ada), Scala, Kotlin, Groovy, Closure, and many others.
- The JVM could be a perfectly good target for a Tiger compiler as well.

For example, the Scala programming language is used by some people to write Android applications, as the Android platform and its tools accept any Java bytecode.

Microsoft .NET

- Microsoft .NET is another framework designed to accommodate several programming languages from scratch.
- It defines a virtual machine whose bytecode has been designed with developers of different languages and compilers.
- Different languages can cooperate easily within the same application, thanks to the common language runtime (CLR) and the well-defined data format.
- The framework class library (FCL) offers services accessible from any language.

Although .NET was initially developed for Microsoft Windows only, it now exists on other systems (Mono for Linux for example).

Is all that new?

Absolutely not.

- In 1978, UCSD (University of California San-Diego) released UCSD p-System and the UCSD Pascal compiler.
- UCSD p-System was a virtual machine *p-Machine* (for "pseudo-machine") running bytecodes called *p-codes*.
- The p-System, and henceforth the UCSD Pascal compiler, could run on all the UCSD microcomputers and minicomputers.

Conclusion

- By targeting a virtual machine, one can write portable programs that will execute on various processors and operating systems.
- Emulating a virtual machine is slow, but an extra compilation phase on the target computer may bring performances close to the ones of natively compiled code.