The Dalvik Virtual Machine

Introduction

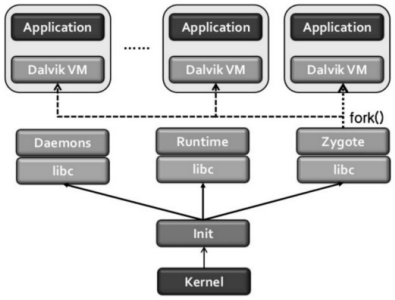
An OS offers a set of common APIs on every platform on which the system runs.
That allows to run the same kind of executable file on different platform, or OS version.
This is possible in a lot of ways, such as cross-compiling the source, or execute it on a Virtual Machine.
In Android, users application are executed trough the Dalvik Virtual Machine.

What does the DVM?

- Executes Dalvik Bytecode.
- Manages multiple instance of itself (Like the JVM does.)
- Provides a "sandbox" where applications can run *without* interact each other.
- Abstracts the problem of memory menagement to the OS.

Applications run on the DVM just like in a "sandbox".

Make it efficient



As mentioned, every application runs on a single instance of the Dalvik Virtual Machine.
Zygote is the process that make the creation of new DVM instace efficient.

1. The Boot-loader loads the kernel and start init process.
2. Starts Zygote process.
3. Initializes a Dalvik VM which preloads and pre-initializes core library classes.
4. The system keeps in an idle state the process until the execution request.
5. Once an application execution request occurs, Zygote forks itself and create new process with the pre-loaded Dalvik VM.

Not a JVM clone

- **The architecture is Register-Based (instead of Stack-Based, like the JVM).**
  Up to 2^16 avaiable registers.

- **DVM uses the Dalvik Bytecode instead of the Java one.**
  Which is far more pragmatic :

```
                      iload s0
add-int d0, s0, s1    iload s1
                      iadd
                      istore d0
```

  The DVM instruction set has 218 opcodes (JVM: 200 opcodes).
  Its use reduces by 30% fewer instructions, but 25% larger code size (bytes) compared to JVM.

- **Constant Pool**
  The dx compiler significantly reduces the size of the costant pool, by inling them directly in the bytecode.
  This permits the use of one single costant pool, unlike the JVM does.

- **It comes without exceptions handling.**

- **Null references**
  DVM does not specify a null type, the 0 value is used instead.

- **Ambiguos primitive types**
  The JVM is strongly typed, and uses different opcodes for every data-type.
  DVM does not distinguish between int/float/null, it uses aget (and aget-wide for double and long).

- **DVM is far more efficient.**
  A Register-Based architecture is up to 47% more efficient than a Stack-Based one!
  The larger code size involves only 1.07% extra real machine loads per VM instruction, which is negligible.