Developing and Benchmarking Native Linux Applications on Android

Leonid Batyuk Aubrey-Derrick Schmidt
Hans-Gunther Schmidt Ahmet Camtepe Sahin Albayrak

DAI-Labor, Technische Universität Berlin

The Second International ICST Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications, 2009

Outline

1. Android Overview
   - What is Android?
   - How does it work?
   - The Dalvik VM
2. Native code for Android
   - Scope
   - Important facts
   - Techniques
3. Benchmarking
   - Performance issues
   - Benchmarking set-up
   - Results
   - Conclusions

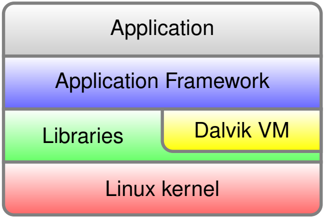
Outline

1. Android Overview
   - What is Android?
   - How does it work?
   - The Dalvik VM
2. Native code for Android
   - Scope
   - Important facts
   - Techniques
3. Benchmarking
   - Performance issues
   - Benchmarking set-up
   - Results
   - Conclusions

What is Android?

- Android is an open-source OS for mobile internet devices
- Android is being driven by the Open Handset Alliance, including Google, HTC, T-Mobile, Samsung, Sony-Ericsson, Motorola and others
- Android is tageted at, but not limited to smartphones. It is supposed for all kinds of mobile devices, including netbooks

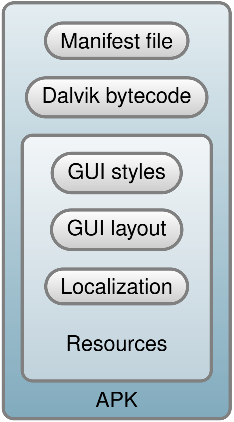
How does it work?



Android comprises of:

- Linux kernel
- Modified BSD libc (bionic)
- Stripped-down unixoid userland
- Custom object oriented IPC (OpenBinder)
- Custom Java VM (Dalvik)

Development of Android applications



- Developers are intended to create applications in Java
- An SDK is provided by Google
  - Emulator
  - Eclipse plugin
  - Debugging utilities
- An application is packaged for distribution in an APK file, which contains:
  - Bytecode
  - Manifest file describing the capabilities etc.
  - Various application resources
- Distribution is possible, but not restricted to, the Android Market.

The Dalvik VM

- Custom Java VM developed by Google
- Uses its own bytecode, not Java bytecode
- Each application runs in its own VM instance for security reasons
- Register-based, optimized for small footprint
- Lacks Just-In-Time compilation and other common optimizations, therefore not performant

Why not speed-up using native code?

Using native code is still not supported, but is expected to become part of the SDK by the end of the year.

Google says:

[...] C/C++ code [...] easily runs 10-100x faster than doing the same thing in a Java loop.

Why not speed-up using native code?

Using native code is still not supported, but is expected to become part of the SDK by the end of the year.

Google says:

[...] C/C++ code [...] easily runs 10-100x faster than doing the same thing in a Java loop.

Outline

1. Android Overview
   - What is Android?
   - How does it work?
   - The Dalvik VM
2. Native code for Android
   - Scope
   - Important facts
   - Techniques
3. Benchmarking
   - Performance issues
   - Benchmarking set-up
   - Results
   - Conclusions

Scope

What is a good reason to use native code?

- Speed up heavy computational tasks
- Time-critical applications
- Running a daemon outside of the application lifecycle

Out of scope:

- 100% native applications are impossible since the UI runs in Dalvik
- Porting big and powerful software like Snort or MySQL is unfeasible due to linking issues

Scope

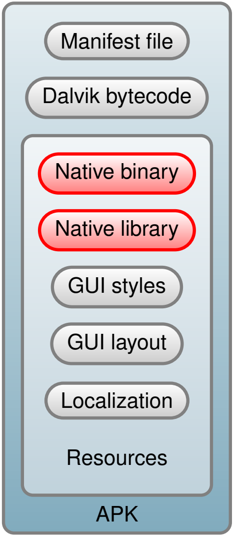
What is a good reason to use native code?

- Speed up heavy computational tasks
- Time-critical applications
- Running a daemon outside of the application lifecycle

Out of scope:

- 100% native applications are impossible since the UI runs in Dalvik
- Porting big and powerful software like Snort or MySQL is unfeasible due to linking issues

Important facts



- Toolchain
  - Code Sourcery G++ (G++-like toolchain)
  - Scratchbox (ARM emulation with a toolchain)
- Different page alignment
  - Dynamic linking becomes difficult
  - Static linking preferred for standalone executables
- Packaging
  - If you want a UI, make your native code a part of an APK
- Size limit
  - Any raw resource which is packaged inside an APK may not exceed 1Mb

Techniques

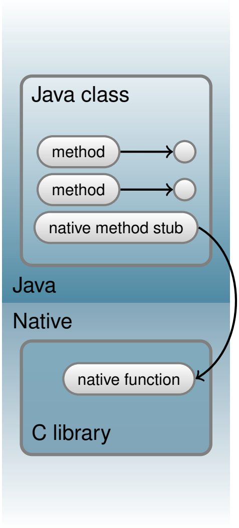
JNI

Java Native Interface

Pipes

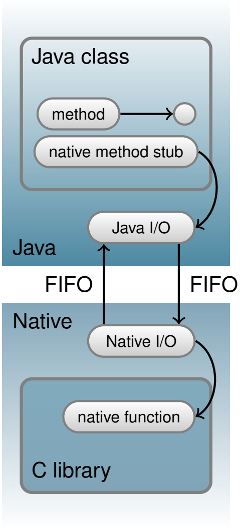
Traditional unixoid IPC via FIFOs

JNI



- JNI - Java Native Interface
- Widely accepted in the Java ecosystem (Eclipse, SWT)
- Widely used in the Android OS implementation
- Currently not supported in the SDK, but planned
- Runs in same thread, no process is being spawned

Pipes



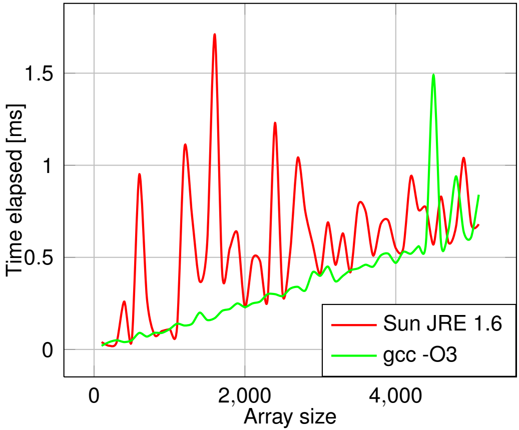
- FIFO - first in, first out
- Widely used for simple IPC on unixoid systems
- Java uses a *named pipe* to communicate to a standalone native executable
- Java I/O is extremely expensive on Android and thus a bottleneck
- Runs in its own thread, can be made a daemon
- This allows us to avoid the standard application lifecycle

Outline

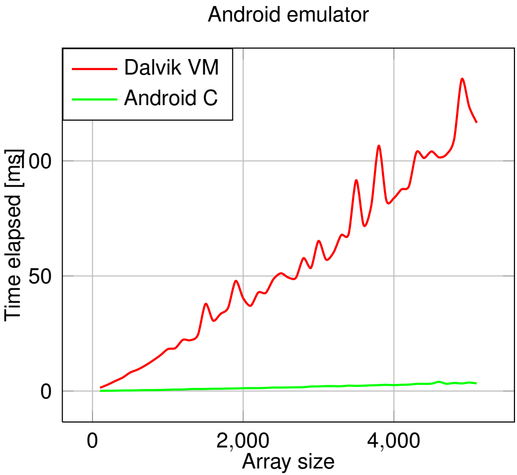
1. Android Overview
   - What is Android?
   - How does it work?
   - The Dalvik VM
2. Native code for Android
   - Scope
   - Important facts
   - Techniques
3. Benchmarking
   - Performance issues
   - Benchmarking set-up
   - Results
   - Conclusions

Performance of the Sun JVM

Linux x86 PC (for comparison)



Performance issues of the Dalvik VM



Performance issues of the Dalvik VM

Dalvik performance problems

- No Just-in-Time compilation
- Optimized for small footprint, not raw performance
- Java I/O (`java.io`) and built-in functions relatively slow

Microbenchmarking approach

- Microbenchmarking focuses on small and uncomplicated benchmarks
- Measuring the performance of the basic computing operations
- Not intended to rate the overall performance of the system
- Not measuring the responsiveness of the UI or the I/O speed

Benchmark set-up

- Heapsort in Java
- Heapsort in a daemon which listenes to a FIFO
- Heapsort in a JNI library
- Built-in Java method for sorting arrays
- Built-in Java method for sorting objects (`PriorityQueue`)
- Quicksort in Java

Setup on Android and on a Linux PC

- Android: Code Sourcery `gcc -O3` vs. Dalvik VM
- Linux: GNU Compiler Collection `gcc -O3` vs. Sun JDK 1.6

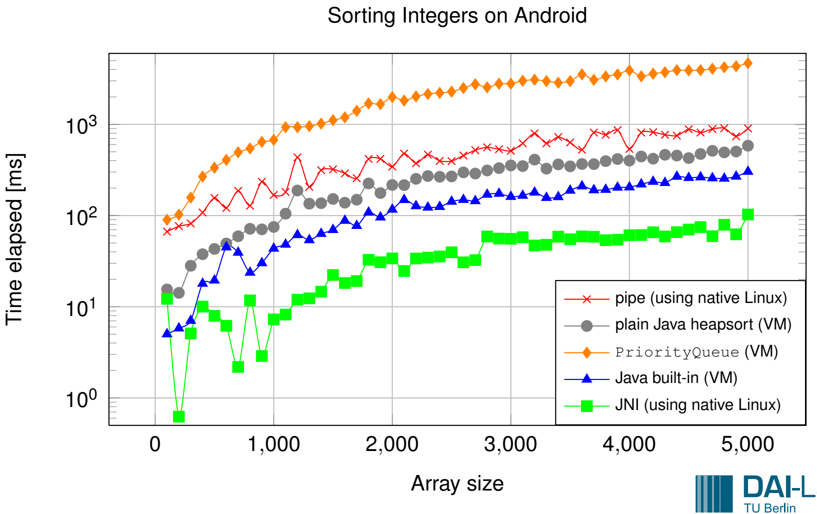
Benchmark set-up

- Heapsort in Java
- Heapsort in a daemon which listenes to a FIFO
- Heapsort in a JNI library
- Built-in Java method for sorting arrays
- Built-in Java method for sorting objects (`PriorityQueue`)
- Quicksort in Java

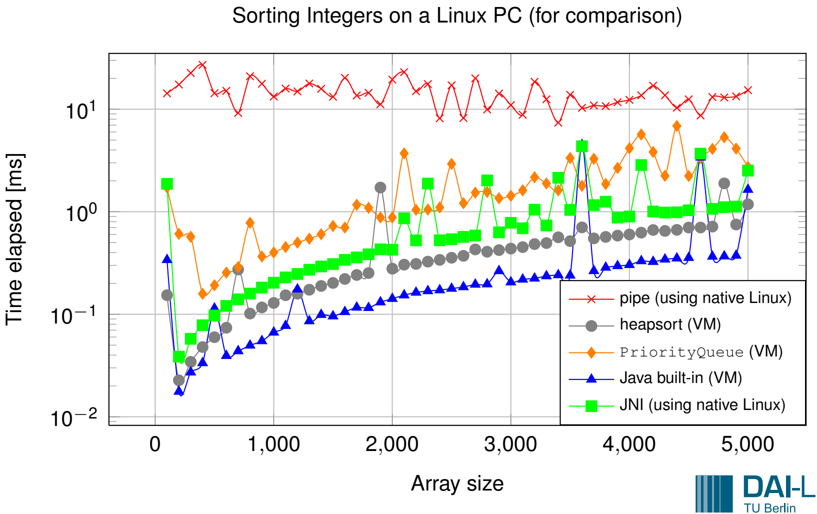
Setup on Android and on a Linux PC

- Android: Code Sourcery `gcc -O3` vs. Dalvik VM
- Linux: GNU Compiler Collection `gcc -O3` vs. Sun JDK 1.6

Results on Android



Results on a Linux system (for comparison)



Conclusions for Android

- JNI is the fastest approach
- JNI is up to 10 times faster than plain Java
- Pipes are unfeasible for data-intensive tasks because of the expensive I/O
- Google should optimize Dalvik:
  - introduce JIT
  - implement computationally complex classpath methods with JNI

Conclusions for Android

- JNI is the fastest approach
- JNI is up to 10 times faster than plain Java
- Pipes are unfeasible for data-intensive tasks because of the expensive I/O
- Google should optimize Dalvik:
  - introduce JIT
  - implement computationally complex classpath methods with JNI

Conclusions for Android

- JNI is the fastest approach
- JNI is up to 10 times faster than plain Java
- Pipes are unfeasible for data-intensive tasks because of the expensive I/O
- Google should optimize Dalvik:
  - introduce JIT
  - implement computationally complex classpath methods with JNI

Conclusions for Android

- JNI is the fastest approach
- JNI is up to 10 times faster than plain Java
- Pipes are unfeasible for data-intensive tasks because of the expensive I/O
- Google should optimize Dalvik:
  - introduce JIT
  - implement computationally complex classpath methods with JNI

Future work

- Port a more common benchmark to Android (maybe LINPACK)
- Benchmark various handsets as they emerge during 2009
- Compare performance of Android to other mobile OSes on the same hardware

Thank you!