POSIX Abstractions in Modern Operating Systems: The Old, the New, and the Missing

Vaggelis Atlidakis, Jeremy Andrus, Roxana Geambasu, Dimitris Mitropoulos, and Jason Nieh

Columbia University
Motivating Example: Execution of iOS Apps on Android
Motivating Example: Execution of iOS Apps on Android

OS abstractions for portable application development!
Motivating Example: Execution of iOS Apps on Android

Initial insight

- Support translation at POSIX level
- UNIX-based systems
- Similar POSIX functionality
Motivating Example: Execution of iOS Apps on Android

Reality

- Cannot implement translation at POSIX level :-(
- iOS, Android platform-specific graphics libraries
Motivating Example: Execution of iOS Apps on Android

**Reality**
- Cannot implement translation at POSIX level :-(
- iOS, Android platform-specific graphics libraries

**Solution**
- Build compatibility at higher-level of abstraction
Study Goals

**Audience:** Developers, researchers, and standard bodies

- Study the evolution of abstractions in modern OSes
- Understand how modern workloads use traditional abstractions
- Identify the needs of modern applications
Study Questions

- Q1: Which POSIX abstractions are unpopular for modern apps?
- Q2: Which POSIX abstractions are popular for modern apps?
- Q3: Is POSIX missing any functionality?
- More in the paper...
Workloads & Methodology

Three Modern OSes
- Android 4.3, Ubuntu 12.04, and OSX 10.10

Client-side Apps
- e.g., Facebook, Twitter, Skype, Chrome, Safari

Common User Workloads
- e.g., post update, tweet, video call, browse
Workloads & Methodology

Static Measurements
- Abstractions linked at large scale
- Analyze native libraries
- Android (>1M apps), Ubuntu (>70K pkgs), OSX (None)
Workloads & Methodology

Static Measurements
- Abstractions linked at large scale
- Analyze native libraries
- Android (>1M apps), Ubuntu (>70K pkgs), OSX (None)

Dynamic Measurements
- Abstractions invoked by common workloads
- Analyze stack traces
- Android (45 apps), Ubuntu (45 apps), OSX (10 apps)
Study Questions

- Q1: Which POSIX abstractions are unpopular for modern apps?
- Q2: Which POSIX abstractions are popular for modern apps?
- Q3: Is POSIX missing any functionality?
Q1: Which POSIX abstractions are unpopular for modern apps?
Q1: Which POSIX abstractions are unpopular for modern apps?

**Few highly linked Interfaces**

Examples
- `memcpy` (99% apps)
- `malloc` (92% apps)
- `memset` (90% apps)
Q1: Which POSIX abstractions are unpopular for modern apps?

**Long tail of unused interfaces**

IPC (only 32% implemented in Android)
- No shared_mem, mq
- Partially pipes, semaphores
- Very few apps link to mkfifo
Study Questions

Q1: Which POSIX abstractions are unpopular for modern apps?

**Long tail of unused interfaces**

FS (76% implemented in Android)
- Missing async I/O functions (aio_*)
- No dbm functions (dbm_*)
- Very few apps link file lock functions
Q1: Which POSIX abstractions are unpopular for modern apps?

- Very few apps link to `mq_*`
- Very few apps link to `aio_*`
Q1: Which POSIX abstractions are unpopular for modern apps?

- Large numbers of unused or unimplemented abstractions
- Departure from traditional IPC and async I/O
• Study Questions
  ○ Q1: Which POSIX abstractions are unpopular for modern apps?
  ○ Q2: Which POSIX abstractions are popular for modern apps?
  ○ Q3: Is POSIX missing any functionality?
Q2: Which POSIX abstractions are popular for modern apps?

**Memory (Examples)**
- `memset`, `memcpy`
- `malloc`, `calloc`
- `mprotect`, `cacheflush`, `setjmp` (JIT)

**Threads (Examples)**
- `pthread_get_specific`
- `pthread_cond_signal`
Q2: Which POSIX abstractions are popular for modern apps?

Percentage of Invocations (45 Android Apps)

- MEM - 68.8
- THREAD - 24.2
- OTHER - 5.0
- FS - 1.2
- IOCTL - 0.6
- IPC - 0.2

Percentage of CPU Time (45 Android Apps)

- MEM - 56.6
- IOCTL - 16.0
- OTHER - 15.2
- FS - 7.9
- THREAD - 3.7
- IPC - 0.6
Q2: Which POSIX abstractions are popular for modern apps?
IOCTL

- Extension API used to shortcut POSIX
- Directly interact with the kernel
- Build functionality not expressed from POSIX APIs
IOCTL

- Analyze stack traces
- Identify libraries heavily invoking ioctl
### IOCTL

- Analyze stack traces
- Identify libraries heavily invoking ioctl

<table>
<thead>
<tr>
<th>OS</th>
<th>1st Library</th>
<th>2nd Library</th>
<th>3rd Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>Graphics (74%) (e.g., libnvrm)</td>
<td>Binder IPC (24%) (e.g., libbinder)</td>
<td>Other (2%)</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>Graphics (52%) (e.g., libgtk)</td>
<td>Network (47%) (e.g., libQtNet)</td>
<td>Other (1%)</td>
</tr>
<tr>
<td>OSX</td>
<td>Network (99%) (e.g., net.dylib)</td>
<td>Loader (1%) (e.g., .dylib)</td>
<td>-</td>
</tr>
</tbody>
</table>

**Top Libraries that Invoke IOCTL in each OS and functionality implemented**
Q2: Which POSIX abstractions are popular for modern apps?

Extension APIs!!!
● Study Questions
  ○ Q1: Which POSIX abstractions are unpopular for modern apps?
  ○ Q2: Which POSIX abstractions are popular for modern apps?
  ○ Q3: Is POSIX missing any functionality?
Graphics

- POSIX omits graphics abstractions
- OpenGL cross-platform API used by applications
- No standard interface to GPUs but ioctl
- Limited extensibility and vendor-specific APIs
IPC

- Binder IPC is a central abstraction in Android
- Android uses ioctl to build Binder in kernel
- Similar patterns in other OSes (MACH IPC, D-Bus)
IPC

- Binder IPC is a central abstraction in Android
- Android uses ioctl to build Binder in kernel
- Similar patterns in other OSes (MACH IPC, D-Bus)

➢ But why not traditional IPC, e.g, pipes?
Benchmarks
- Measure latency of transactions
- Binder benchmark from Android source
- MACH using MPMMTest from XNU

Consumer Devices
- Nexus-7, MacBook Air, Dell XPS
Limitations of traditional IPC

- Similar scalability issues across the three OSes
- High-latency for large transaction sizes
Benefits of new IPC

- Perform with near-constant latency
- Leverage in-kernel single- and zero-copy mechanisms
Threads

- GUI apps require low-latency UI threads
- Dispatching events is the new paradigm
- High-level event and thread management APIs
  - Android: ThreadPool and EventLoop
  - Ubuntu: ThreadPool and EventLoop
  - OS X: Grand Central Dispatch
Q3: Is POSIX missing any functionality?

- Graphics support
- New IPC mechanisms
- Threading APIs for event-driven programming
Evolution of systems and applications

In the past

OS abstractions for portable application development
In the past

```
App  App  App

POSIX API

OS
```

... the major good idea with UNIX was its clean and simple interface: open, read, and write"

~K. Thompson. Unix and Beyond, 1999
Evolution of systems and applications

In the past

OS abstractions for portable application development

Now

Multiple layers of platform-specific software

Software Framework

Extension APIs

POSIX API
Contributions & Future Work

- Tools and methodology for static and dynamic analysis
- Identified **popular**, **unpopular**, and **missing** POSIX abstractions
- Open sourced tools and data:
  - [https://columbia.github.io/libtrack/](https://columbia.github.io/libtrack/)
Contributions & Future Work

- Tools and methodology for static and dynamic analysis
- Identified **popular**, **unpopular**, and **missing** POSIX abstractions
- Open sourced tools and data:
  - [https://columbia.github.io/libtrack/](https://columbia.github.io/libtrack/)
- Revisit OS abstractions for IPC, Threads, and Graphics