A Study of Modern Linux API Usage and Compatibility:

WHAT TO SUPPORT WHEN YOU'RE SUPPORTING

Chia-Che Tsai
Bhushan Jain
Nafees Ahmed
Abdul Donald E. Porter
OSCAR Lab
Stony Brook University
System Building: When You Become a Parent

Our experience from building a OS with Linux API support (Graphene library OS [Eurosys’14]):

March 2011
Project started

September 2012
12 syscalls supported
hello world

October 2013
131 syscalls supported
apache
gcc
makefile
etc.

When can we claim having a decent system?

API compatibility is measured as all-or-nothing (impractical for system developers)
What to Expect from This Paper:

• A **method** to quantify properties of API support:
  • From importance of APIs to completeness of systems
  • Practical, generalizable to other OSes

• A **study** on modern Linux APIs:
  • Including different API types (e.g., syscalls, ioctl opcodes)
  • How Linux users rely on Linux APIs
  • An optimal path to build a Linux-compatible system
Chapter 1
How to Measure API Usage and Compatibility
First Thought: # of APIs or Applications

systems  
JohnnyOS  
support 2 APIs or 1 app  
EmmaOS  
support 2 APIs or 2 apps

APIs (ex: syscalls)  
sys_ladder()  
sys_steer()  
sys_lift()

applications  
fire-truck.app  
crane-truck.app  
lifter.app

Can we conclude who has better API compatibility?  
(No, we cannot)
Taking Popularity into Consideration

APIs are not equally popular (e.g., `sys_read` > `sys_sync`)

Neither are applications (e.g., Bash > CVS)

New metrics to reflect both users and app developers’ choices
We Need 2 Metrics for Building API Support

• Which APIs should I implement first?
  **API Importance**
  (API usage)

• What is the progress of API support in my system?
  **Weighted Completeness**
  (system’s API compatibility)
A Metric for APIs: API Importance

API importance = Probability that a random user installs any applications using the API

\[
\text{Pr} \left[ \text{crane-truck.app is installed or fire-truck.app is installed} \right] \leq 1 - (1 - 60\%) (1 - 80\%) = 92\%
\]

(upper bound)

If the API is missing, how many users will complain?
A Metric for Systems: Weighted Completeness

Weighted completeness =

Fraction of installed applications to be supported by the system, for a random user

\[
\approx \frac{(0.6 + 0.8)}{5} = 0.28 = 28\%
\]

If a user switches to the new system, how many apps will still work?
Quick Summary

• **API Importance (for each API):**
  % of users that install any apps using the APIs

• **Weighted Completeness (for the whole system):**
  % of a user’s installed apps supported by the system
Chapter 2
A Study of Linux APIs
and How It Can Help API Support
A Large-Scale Linux API Study

- Applications Sample: Ubuntu 15.04 official repositories
  66,275 ELF binaries in 22,459 amd64 packages
  - EXEs linked with LIBs
    - 48% shared LIBs
    - 52% EXEs

- Installation statistics: Popularity Contest
  - Ubuntu: 2.7 million installations (http://popcon.ubuntu.com)
  - Debian: 0.2 million installations (http://popcon.debian.org)

A large, representative sample to draw meaningful observations
Tons that You Can Find in the Study

• For researchers: (in the paper)
  • Observations to motivate ideas

• For maintainers: (in the paper)
  • Evidences to justify or guide decisions

• For builders:
  • Rationale for prioritizing APIs to implement
  • Quantifying system building goals
Prioritizing Linux System Calls

224 are used by at least one app for each user
Ex: read, exit, clone

45 used by < 10%
Ex: ustat, tee, getcpu

6 completely unused
Ex: get_robust_list, mq_notify, move_pages

Even if importance is ~100%, ranking is meaningful for prioritizing APIs to support
Using API Importance As Heuristic

Higher-ranking APIs are likely to support more applications for a user.

- First 40 syscalls: used by every packages (must implement first).
- Last 75 syscalls: used by very few packages (e.g., setdomainname() by hostname).

Ideal for prioritizing APIs to maximize weighted completeness.
Evaluating the System while Building It

- Goal: maximize weighted completeness
- Approach: implement the most important APIs (syscalls) first

More nearly optimal path than only relying on developers’ intuition
More in the Paper

• More API types:
  • Opcodes of vectored syscalls (e.g., ioctl, fcntl, prctl)
  • Pseudo-files (e.g., /proc, /dev, /sys)
  • Library functions (e.g., GNU library C)

• More systems: e.g., L4Linux, User-Mode-Linux, libc variants

• Hints for Maintainers:
  • When is the timing of deprecation?
  • Where is the sweet spot of limiting APIs (e.g., for security)?
  • What is app developers’ preference?
Tool, Data and Code Available Soon!

www.oscar.cs.stonybrook.edu/api-compat-study

Online Evaluation Tool

Data Set (2.6 M records) for Download
Conclusions

• An API study that reassuringly answers the questions of system developers, from planning stage to release.
  • Encourage builders with better methods to strategize/evaluate.
  • Motivate researchers and justify maintainers’ decisions.

• Lessons for evaluating all-or-nothing properties
  Analysis techniques (e.g., binary analysis)
  + User studies (e.g., application popularity)

Tool / Data / Code: www.oscar.cs.stonybrook.edu/api-compat-study

Chia-Che Tsai
chitsai@cs.stonybrook.edu