olibc: Another C Library optimized for embedded Linux

Jim Huang (黃敬群) <jserv@0xlab.org>
Developer, 0xlab

Feb 22, 2013 / Embedded Linux Conference
Rights to copy

© Copyright 2013 0xlab
http://0xlab.org/
contact@0xlab.org

Attribution – ShareAlike 3.0

You are free

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions

**Attribution.** You must give the original author credit.

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: http://creativecommons.org/licenses/by-sa/3.0/legalcode

Corrections, suggestions, contributions and translations are welcome!

Latest update: Feb 22, 2013
What I will discuss about...

- I learned a bit about the toolchain and system library optimizations while developing Android based projects.
- It might be a good idea to “reuse” them in ordinary embedded Linux projects.
- Therefore, I plan to emphasize on how to leverage the engineering efforts originally from Android world.
  - Review C library characteristics
  - Toolchain optimizations
  - Build configurable runtime
  - Performance evaluation
Related talks

- **Android Builders Summit 2013**
  - LLVMLinux: Compiling Android with LLVM - Behan Webster, Converse in Code - Behan Webster

- **Embedded Linux Conference 2013**
  - Toybox: Writing a new Linux Command Line from Scratch - Rob Landley
  - System-wide Memory Management without Swap - Howard Cochran
  - Bringing kconfig to EGLIBC - Khem Raj
Agenda

(1) Take from Android
(2) olibc: configurable
(3) Optimizations
(4) Action Items
Take from Android

bionic libc, dynamic linker, debugging facilities
We know, Android is not Linux, but...

We're __taking__ something useful back to embedded Linux.
from Rob Landley's talk

- "Dalvik is the new ROM basic"
- ...
- "why not extend toolbox/bionic instead of replace? – just enough to run dalvik. (The new ROM BASIC.)"

Our "something useful" is the base to launch Dalvik Virtual Machine and the above Android Framework

Source: http://www.landley.net/talks/celf-2013.txt
Why build bionic-derived libc?

- **License**
  - glibc (LGPL), uClibc (LGPL), dietlibc (GPL), musl (MIT)

- **Optimized for major (consumer) targets**
  - ARMv7 + MIPS + Intel Atom optimizations
  - glibc (good arm/mips/atom), uClibc (simpler arm/mips/x86), dietlibc (N/A), musl (simple x86)

- **API coverage is getting more complete by versions.**

- **Catch up with latest SoC technologies**
  - Contributors: Google, Intel, Qualcomm, Texas Instruments, NVIDIA, ST-Ericsson, Linaro, MIPS, etc.

- **The problem is, Android is not a community project.**

“Copyleft is dying. GPLv2 was category killer, synonymous with copyleft.” – Rob Landley
Goals of olibc

• Create small, fast, free, and standard-compliant implementation for C Library.
• Offer configurable levels of functionality and should scale from tiny embedded Linux up to general purpose environments such as Android-powered smart devices.
• Provide system utilities around C library  
  – benchmarking, validation, prelinker, ...
• Introduce steady security, performance, and conformance fixes.
# include <stdio.h>
int main() {
    printf("Hello Wolrd\n");
    return 0;
}

Let's review the programming model...
Programming Model (multi-threaded)

**Bare-metal**

- "Stack" Section
- "BSS" Section
- "Data" Section
- "Text" Section

**Multi-Thread**

- Stack
- Stack
- Stack
- BSS
- "Data" Section
- Text
- Text
- Text
- scheduler
Programming Model (multi-process)

**Multi-Process [Linux]**

- application
- application
- application
- Library
- kernel
- scheduler

**Multi-Process [Android]**

- application
- application
- application
- Framework & VM
- middleware & Library
- kernel
- scheduler


bionic libc

- Small C Library implementation
  - mixture of NetBSD (libc) and FreeBSD (libm)
- BSD license
- Partially POSIX compatible; not compatible with glibc
- No SysV IPC support (due to Binder IPC)
- Support for ARM (w/ Thumb), x86, MIPS
- Fast pthread implementation (based on futexes)
- Small footprint

<table>
<thead>
<tr>
<th>Library</th>
<th>Version</th>
<th>Path</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>glibc</td>
<td>2.11</td>
<td>/lib/libc.so</td>
<td>1,208,224 bytes</td>
</tr>
<tr>
<td>uClibc</td>
<td>0.9.30</td>
<td>/lib/libuClibc.so</td>
<td>424,235  bytes</td>
</tr>
<tr>
<td>bionic</td>
<td>2.1</td>
<td>/system/lib/libc.so</td>
<td>243,948  bytes</td>
</tr>
</tbody>
</table>
Not in bionic libc

- Complete wide chars
- C++ exceptions (limited since NDKr5)
- Full C++ STL
- Full POSIX Thread
Memory Map [Android pre-4.x]

- **app_process**
- **Java apps and resource data**
  - *.apk,*.jar,*.ttf etc.
- **Shared lib; libc, libwecore etc.**
  - various .so files
- **/system/bin/linker**
- **Stack**

No memory with execution attribute

File mapping direction

Some memory are of execution attribute

File mapping direction
Memory related changes

- **ASLR (Address space layout randomization) since Android 4.0**
  - help protect system and third party applications from exploits due to memory-management issues
  - PIE (Position Independent Executable) is added since Android 4.1
    - original ELF prelinker was removed

- **AddressSanitizer since 4.1**
# AddressSanitizer vs. Valgrind

<table>
<thead>
<tr>
<th></th>
<th>Valgrind</th>
<th>AddressSanitizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap out-of-bounds</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Stack out-of-bounds</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Global out-of-bounds</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Use-after-free</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Use-after-return</td>
<td>No</td>
<td>Sometimes/Yes</td>
</tr>
<tr>
<td>Uninitialized reads</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Overhead</td>
<td>10x-30x</td>
<td>1.5x-3x</td>
</tr>
<tr>
<td>Host platform</td>
<td>Linux, Mac OS X</td>
<td>where (latest) GCC/LLVM runs</td>
</tr>
</tbody>
</table>
==7161== ERROR: AddressSanitizer global-buffer-overflow on address 0x2a002194 at pc 0x2a00051b bp 0xbbeeaf0c sp 0xbbeeaf08
READ of size 4 at 0x2a002194 thread T0
    #0 0x40022a4b (/system/lib/libasan_preload.so+0x8a4b)
    #1 0x40023e77 (/system/lib/libasan_preload.so+0x9e77)
    #2 0x4001c98f (/system/lib/libasan_preload.so+0x298f)
    #3 0x2a000519 (/system/bin/global-out-of-bounds+0x519)
    #4 0x4114371d (/system/lib/libc.so+0x1271d)
0x2a0002194 is located 4 bytes to the right of global variable 'global_array (external/test/global-out-of-bounds.cpp)' (0x2a002000) of size 400
Shadow byte and word:
  0x05400432: f9
  0x05400430: 00 00 f9 f9
More shadow bytes:
  0x05400420: 00 00 00 00
  0x05400424: 00 00 00 00
  0x05400428: 00 00 00 00
  0x0540042c: 00 00 00 00
  =>0x05400430: 00 00 f9 f9
  0x05400434: f9 f9 f9 f9
  0x05400438: 00 00 00 00
  0x0540043c: 00 00 00 00
  0x05400440: 00 00 00 00
Stats: 0M malloced (0M for red zones) by 35 calls
Stats: 0M realloced by 0 calls

```c
int global_array[100] = {-1};
int main(int argc, char **argv) {
    return global_array[argc + 100];  /* BOOM */
}
```
Shared library issues

- Older Android dynamic linker has an arbitrary low (for larger applications such as GStreamer, LibreOffice) limit on number of shared libs: 128
  - Until Sep 12, 2012, dynamically allocating soinfo-structs in linker is implemented.

- Mozilla: Use a hacked-up copy of the bionic linker in our own code, in addition to the system one.
  - two run-time linkers not aware of each others ended up a failure
## C++ Integrations

<table>
<thead>
<tr>
<th></th>
<th>C++ Exception</th>
<th>C++ RTTI</th>
<th>Standard Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>gabi++</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>stlport</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>gnustl</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- olibc provides stlport, which depends on wchar support in libc.
Debuggerd

- Nice embedded-specific crash handler
  - used on all Android devices

- Crash report data placed in log/tombstone

- Debuggerd also facilitates connecting debugger to dying process
  - Can halt and wait for gdb to attach to the process

- Apache license
How Debuggerd works

- Debuggerd acts as a crash-handling daemon
- Adds default signal handler to each process, which handles any signals that generate core
  – included in bionic, thus every application gets it
- Signal handler captures deadly signal and contacts debuggerd
- Debuggerd records information using ptrace (registers, stack, memory areas), and /proc
- Has built-in ARM stack unwinder for generating a backtrace
- Automatically rotates a fixed number of crash reports

Reference:
https://wiki.linaro.org/WorkingGroups/ToolChain/Outputs/LibunwindDebuggerd
• Unwinding = processing stack and memory image to create a backtrace
• Backtrace is very compact - summarizes stack information nicely
• Local variables usually not available
• Different methods available, depending on compilation flags
Crash Handler

- New crash handler written by Tim Bird of Sony
  - Based on debuggerd from Android
- Implemented as a core file handler
- Writes crash report to a “tombstone_0x” file in /tmp/tombstones
- Writes information from /proc, ptrace, and kernel log buffer
- Also writes some information to the kernel log
- Information: http://elinux.org/Crash_handler
License Issue

  - Google tries to “clean” Linux kernel headers to avoid the GPL
olibc: configurable
Major problem: broken toolchain
How Toolchain works

Source (.c and .h)

Preprocessed source (.i or .ii)

Asm (.s)

Relocatable (.o)

Binary (Executable)

gcc --save-temp

gcc

cpp

cc1

as

collect2

library

CRT*

link script

Id (Soft-Link)

ld.gold

ld.bfd

gcc

binutils

glibc
External Toolchain Issues

• CodeSourcery Toolchain doesn't use gold linker, and Android's original build flags are a bit aggressive.
  – e.g. ICF (Identical Code Folding), which is gold only redundancy elimination
  – Option: `--icf` (known to bring 5% size reduction)

• Default link script doesn't take olibc into consideration.

• Sometimes, toolchains have optimization bugs
Build Android compatible toolchain

- **Barebone-style building:**
  - Inside Android tree
  - Specify all system and bionic header file paths, shared library, paths, libgcc.a, crtbegin_*.o, crtend_*.o, etc.

- **Standalone-style building:**
  - Convenient for native developers:
    - `arm-xxx-eabi-gcc -mandroid --sysroot=<path-to-sysroot> hello.c -o hello`
      
      (<path to sysroot> is a pre-compiled copy of Bionic)
olibc: Configurable and Optimized

- Configured using the Kconfig language pioneered by the Linux kernel
  - Extensions, Library settings, crash handler, ...

- Encapulate the Android build system to become simpler and dedicated one.

- Allow full optimization techniques originally applied by Android including implementation and toolchain
  - SoC enhancements

- Use repo to manage source tree
  - repo init -u https://github.com/olibc/manifest.git
  - repo sync
Optimizations
1. Goal: Visibility of a function should match the API spec in programmer’s design.

2. Solution:
First, systematically applying the 5 steps. Fundamentally, need to go through the APIs of each library:

- Consciously decide what should be “public” and what shouldn’t.

3. Result: ~500 KB savings for opencore libs

4. Key: The whole hidden functions can be garbage collected if unused locally:

5. Toolchain’s options:
-ffunction-sections,
-Wl,--gcsections,
Build Tweaks: code size

- Android default inline options:
  -finline-functions
  -fno-inline-functions-called-once

Use genetic algorithm to find best-fit:
- finline
- fno-inline-functions
- finline-functions-called-once
--param max-inline-insns-auto=62
--param inline-unit-growth=0
--param large-unit-insns=0
--param inline-call-cost=4

<table>
<thead>
<tr>
<th></th>
<th>GCC-4.2.1</th>
<th>GCC-4.4.3</th>
<th>GCC-4.4.3 (tuned inline options)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native system image</td>
<td>23,839,291</td>
<td>23,027,032</td>
<td>22,087,436</td>
</tr>
</tbody>
</table>

Source: Smaller and Faster Android, Shih-wei Liao, Google Inc.
Instrumentation-based FDO (Feedback-Directed Optimization)

1. Build twice.
2. Find representative input
3. Instrumentation run: 2~3X slower but this perturbation is OK, because threading in Android is not that time sensitive
4. 1 profile per file, dumped at application exit.

```
arm-xxx-eabi-gcc -fprofile-generate=./profile
```

```
arm-xxx-eabi-gcc -fprofile-use=./profile.zip
```

Optimized Binary

Optimized Binary with FDO

Instrumented Binary

Profile.zip

Run the instrumented binary

Representative Input Data
Global hotness for ARM (HOT_BB_COUNT_FRACTION, Branch prediction routine for the GNU compiler, gcc-4.4.x/gcc/predict.c)
- 1% improvement on Android's skia library as belows.
- smaller effects on smaller Android benchmarks.

<table>
<thead>
<tr>
<th>Content</th>
<th>Work</th>
<th>default</th>
<th>fdo-default</th>
<th>fdo-modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of libskia</td>
<td>default</td>
<td>7,879,646</td>
<td>7,396,032</td>
<td>7,319,668</td>
</tr>
<tr>
<td>Size reduction</td>
<td>0.00%</td>
<td>6.14%</td>
<td>7.11%</td>
<td></td>
</tr>
<tr>
<td>Stdev (over 100 runs)</td>
<td>0.28</td>
<td>0.63</td>
<td>0.26</td>
<td></td>
</tr>
<tr>
<td>Speedup</td>
<td>1</td>
<td>0.98</td>
<td>0.97</td>
<td></td>
</tr>
</tbody>
</table>

Source: Smaller and Faster Android, Shih-wei Liao, Google Inc.
Dynamic Linker Optimizations

• Why?
  – The major reason to optimize dynamic linker is to speed up application startup time.

• How?
  – Implement GNU style hash support for bionic linker
  – Prelinker improvements: incremental global prelinking
(normalized) Dynamic Link time

![Bar graph showing normalized dynamic link time for various processes](image)
(normalized) Symbol Lookup number
DT_GNU_HASH: visible dynamic linking improvement =
Better hash function (few collisions)
+ Drop unnecessary entry from hash
+ Bloom filter

```
void foo (){  
printf("fooooo");  
bar();  
}
```
## GNU-style Hash

<table>
<thead>
<tr>
<th>name</th>
<th>2sym collision #</th>
<th>3sym collision #</th>
<th>3+sym collision #</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysv</td>
<td>1749</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>libiberty</td>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dcache</td>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>djb2</td>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sdbm</td>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>djb3</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rot</td>
<td>337</td>
<td>39</td>
<td>61</td>
</tr>
<tr>
<td>sax</td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fnv</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oat</td>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Experiment by Jakub Jelinek
<table>
<thead>
<tr>
<th>Symbols in ELF</th>
<th>lookup #</th>
<th>fail#</th>
<th>gnu hash</th>
<th>filtered by bloom</th>
</tr>
</thead>
<tbody>
<tr>
<td>gnu.gp</td>
<td>3758</td>
<td>23702</td>
<td>19950</td>
<td>23310</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18234(78%)</td>
</tr>
<tr>
<td>gnu.gp.re</td>
<td>3758</td>
<td>20544</td>
<td>16792</td>
<td>19604</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14752(75%)</td>
</tr>
<tr>
<td>gnu.lp</td>
<td>61750</td>
<td>460996</td>
<td>399252</td>
<td>450074</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>345032(76%)</td>
</tr>
<tr>
<td>gnu.lp.re</td>
<td>61750</td>
<td>481626</td>
<td>419882</td>
<td>448492</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>342378(76%)</td>
</tr>
</tbody>
</table>

NOTE: Android 4.0 removes the support of prelinker, but gnu style hash is still useful.

H = \{x, y, z\} = hash functions
Hash function may collision
→ Bloom filter may got false positives
Action Items
- Use eglibc-like Option Group
  - based on POSIX.1-2001 specifications
- Comply with relevant standards
- Resolve external toolchain issues
  - Allow arm-none-eabi- and arm-none-linux-gnueabi-
  - Don't depend on prebuilt toolchain anymore
- Collaboration: crosstool-ng, buildroot, yocto, ...
- Validation: improve unit-test, bench, ABI test
- More SoC enhancements
- Extensions
  - BioMP: Migrating OpenMP into Bionic
    http://code.google.com/p/biomp/
• olibc hosted at GitHub: http://olibc.github.com/
• Optimizing Android Performance with GCC Compiler, Geunsik Lim
• Embedded-Appropriate Crash Handling in Linux, Tim Bird
• Smaller and Faster Android, Shih-wei Liao, Google Inc.