Faculty Industrial sciences & technology
1st Master MIT

Academic year 2013–2014

Bionic vs Glibc report
Master thesis

Mathieu Devos

Promotors: S. Vrijders
D. Staessens
K. Casier

Abstract

The goal of this report is to find and point out the restrictions that apply when using the Bionic library compared to the standard glibc. These are both C/C++ standard libraries. The two libraries that will be researched are:

- *GNU C Library (glibc)*
- *Bionic Library (bionic)*

Both these standard libraries provide support for the C and C++ language but are used on different platforms. Where glibc is used within Linux distributions, bionic is used on android based systems. Over the course of this report we will try to provide insight in the Bionic library. We will try to find the origin and use of the bionic library and why glibc was not adequate enough for the job. After that we will also provide the restrictions and traps that come with the use of the Bionic library. Concluding is done with the general information and where to find information before coding with the bionic library.

In general the standard C/C++ library is small compared to other languages and should provide adequate speed when using low level programming. Since the standard library is so small it is very easy to port it to new platforms. This raises the question why Google decided to not use glibc but write their own adaptation of a standard C/C++ library. In this report we hope to find the answer to that question, while providing details where both libraries differ.

Table of contents

Chapter 1

Origin of bionic

Bionic is an adaptation of the BSD[1] standard C library. It is used on the android platform and replaces glibc. Since Android is based on linux it would be logical to assume that they used the same C library, however due to the specifications of Android a new library was used. The word *bionic* means "having artificial parts", this directly shows that the bionic library consists of different parts: elements from glibc from linux, from the BSD library and uniquely written pieces of code specific for the bionic library. The reason for this assembly is because of the earlier mentioned android specifications. Since android is designed to run on mobile devices it tends to have several characteristics. Some of these characteristics are what determined to use bionic as the standard C/C++ library in these devices. The most profound characteristics are:

- Limited space and thus limited storage
- Lower CPU speed (around 30-50% lower compared to notebooks and desktops)
- Not fully open source

What exactly is "not fully open source" in this context? While the kernel itself is based on linux and thus has to use the GPL[2]. However it was a goal for android to create an operating system that allows third party applications. Not only could people charge money for these applications, they wanted it so that the code of these third party apps didn't fall under the GNU Public License.
A solution was found within the BSD licenses. These licenses do not fall under the copyleft rule and thus can have copyrights. This allows android programmers to not be

[1] Berkely Software Distribution - A UNIX based operating system
[2] GNU General Public License - A free software license

forced into open source when coding their software.
However, a note should be made here. Linus Torvalds stated that user space programs are not covered by the kernel license. This means that even if bionic (part of the kernel) would fall under the GPL license it would cause no effect for applications running in userspace. This leads to the conclusion that with the use of the BSD license android wants to grant control to others (developers, producers, ...) and provide a choice when it comes to licensing.

Due to limited storage and cpu speeds an adaptation was required before google could implement the library into android. Parts for this library were found in both glibc and in the BSD library. Not only do those items factor into this decision but also faults in glibc such as buffer overflows, unsafe handling of strings and memory functions. Due to these factors bionic is often regarded a non-standard C library.

Chapter 2

Advantages of bionic

A first advantage is the optimization of code due to the removal of all comments from the header files. This reduces the size of all those headers and helps with a speedy process in a storage-constricted space.

Second advantage is that the GPL is kept out of user-space. This was not really an issue but with glibc moving towards LGPLv3 it might become an issue. The use of the BSD license prevents this as a whole and keeps the GPL license out of the userspace in android.

As stated earlier, since android is running on space limited devices it helps immensely that bionic is around 200 kilobytes. Compared to glibc that is around 400 kilobytes. This is done by leaving out bloated code and reducing unneeded parts or even removing them from the library.

A last advantage for bionic can be found with the optimization for slower CPU(s). This means that changes were made to the implementation of pthread for lower clockspeeds. While this is an advantage for bionic, it can be seen as a disadvantage compared to other libraries as this library is specifically designed for lower CPU speeds.

Chapter 3

Restrictions and disadvantages of bionic

In this chapter we will elaborate on the restrictions that working with bionic raises and the disadvantages of using this C library.

3.1 C++

Due to the limited size of bionic most of the restrictions apply to the higher order programming languages, most noticeable with C++.
A part of this information can be found in the CAVEATS file of the bionic library: "libc/CAVEATS".

3.1.1 Exceptions

C++ exceptions are not supported in the bionic library. This is because android feels that on embedded systems these lead to larger and slower code without enough use when actually implementing these exceptions.

An example of this can be deduced directly from the CAVEATS file:

> "even when so-called zero-cost exception schemes are implemented, they enforce very large numbers of registers spills to the stack, even in functions that do not throw an exception themselves"

The reason for this can be found with Java, the main programming language for android. Because Java handles all exceptions in a runtime package it is not needed for bionic to support C++ exceptions and make the entire library larger and slower.

3.1.2 STL

Bionic does not include a C++ STL[1]. Again due to size constraints a part of an often included library was left out of Bionic. This means that if designers, programmers, developers, ... want to have access to either one of these functions:

- Algorithms
- Iterators
- Containers
- Functional

They will have to add an STL manually through other libraries, such as: SGI[2]. The reasons why this library is not included is manifold.

- Big, unreadable, bloated error messages
- Opens up a lot of traps for developers
  - Templates can lead to long, slow code
  - Initiation of templates increases compilation time
  - Initiation of templates increases memory usage
  - Containers should not be used as base classes
  - ...
- Large library as a whole, upwards of 100KB (bionic is only ~200KB)

3.1.3 Overview

This concludes the restrictions that apply for C++ programmers. Do note however that for almost every one of these restrictions there are workarounds available. The bionic library does not choose to implement these added functions for previously stated reasons and most likely will never implement these.

3.2 Pthreads

Pthreads stand for POSIX[3] threads. These are standardized threads defined by IEEE and are compatible across operating systems.

[1] Standard Template Library - library that provides algorithms, containers, functions and iterators
[2] `http://www.sgi.com/tech/stl/`
[3] Portable Operating System Interface

3.2.1 Cancellation

Bionic does not support Pthread cancellation. When coding multithreaded programs correctly it should be noted that the use of this function is unnecessarily. Because of this, leaving this function out of the library reduces it's size, complexity and bloat.

3.2.2 pthread_once()

This function comes without C++ execptions thrown from the init function, or the init function doing a fork().

3.2.3 pthread_atfork()

The function is simply unavailable in the bionic library and should be avoided during code development.

3.2.4 Overview

The libpthread, containing all the pthread header files, comes with some limitations but could be used by coders who which to use these header files. Special attention should be payed towards the earlier mentioned exceptions in this collection of header files.

3.3 Miscellaneous

Other exceptions and restrictions that apply to the bionic library will be noted here. The goal of this is to provide a total overview of what territory comes with the use of the bionic library.

3.3.1 Locales and Wide characters

These two are unsupported in the library and should be avoided altogether. Android recommends the use of ICU for everything related to i18n, this basically means that as a coder you should be using Unicode.

3.3.2 User-account-related functions

Functions that relate to user-accounts are currently implemented but as pseudo-functions. For example *getpwd* will return to root. These functions, however, should later on be

implemented. This is due to the reason that Android does not provide support for distinct users on the filesystem. When this has been implemented, one could expect these functions to be added soon after.

3.3.3 ABI bugs

ABI[4] bugs can be found in the root folder of the library and should always be carefully examined before working with the library. These can be found in the file *ABI-bugs.txt*.

Current bugs (at the time of writing) are:

- time_t is 32 bit
- off_t is 32 bit (off64_t is present but no _FILE_OFFSET_BITS support)
- sigset_t is too small on ARM (android devices) and x86 (but correct on MIPS), so support for real-time signals is broken
- Too few TLS slots leading to allocation shortage for pthread_key_t (should be 128, but can't be reached)
- atexit(3) handlers registered by a shared library aren't called on dlclose(3). Only affects ARM.

[4]Application Binary Interface

Chapter 4

Conclusion

We can conclude that the bionic library is a very lightweight library and comes with certain restrictions. While some of these restrictions are very strict, such as the C++ restrictions, others can be worked around.

It is advised before starting to write code that implements the bionic library to read through the *readme*[1] and the *CAVEATS*[2] file.
Both of these files can be found in the git from the bionic library:

```
https://github.com/android/platform_bionic
```

The final conclusion can be drawn that while the bionic library does have its shortcomings it does provide a small, lightweight standard C and C++ library.

[1] platform_bionic/libc/kernel/README.TXT
[2] platform_bionic/libc/CAVEATS