powerpc64 architecture support in FreeBSD ports

Piotr Kubaj, Bsc

1. Abstract

IBM POWER processors are 64-bit CPU's designed primarily for server market. With POWER9, there has been renewed interest in them, due to the use of open-source firmware and focus on security and control of the hardware. There are also new desktop boards with POWER9. Because of that, support for them has been recently greatly improved in FreeBSD with increased driver compatibility and more 3rd party software having available. For my project, I build the whole ports tree using Poudriere and fix the compilation errors I meet. In this paper, I specify challenges met during porting software to work on POWER processors on FreeBSD and show how most problems can be solved. FreeBSD on POWER architecture runs in big-endian variant only and uses old toolchain – with GCC 4.2 and binutils 2.17. This is why many problems are related to fixing bugs in big-endian variants of code and solving issues related to the old toolchain that the operating system uses.

Keywords: freebsd, powerpc64, ibm power, ports, endianness, big endian, little endian.

2. Introduction

2.1. POWER architecture

This paper describes the effort for running FreeBSD ports on powerpc64 architecture. IBM POWER are 64-bit RISC processors designed specifically for the server market, although there are also boards for desktops and workstations using those processors.

Before POWER9, last POWER processors released for desktop computers, were PowerPC 970MP (based on POWER4) available in Apple PowerMacs G5.[1]

With the release of POWER9, there appeared new desktop computers with POWER processors, making this architecture again interesting for desktop users.[2]

IBM also made the firmware fully free and open source, making this platform owner-controllable, in contrary to other CPU vendors. This aspect makes it more appealing to some users than its competitors.[3]

2.2. Endianness

Endianness is the order in which bytes are read in larger numerical values.
In big-endian architectures, the most significant byte is stored the first (has the lowest address). This is similar to the order used commonly by people.

Little-endian architectures are the opposite. The first number is the least significant byte. The most significant byte is the last one in a given number and has the highest addres. This is in contrary to number ordering used in everyday life.[4]

POWER8 and POWER9 processors are bi-endian – they can run in both modes. However, older POWER CPU's, like PowerPC used in Apple Macintoshes, can only run in big-endian mode (there are some older POWER CPU's able to run little-endian mode but they are not common). Because of that, FreeBSD currently only supports big-endian mode. It is also important that some buses may be little-endian, even though the CPU is big-endian – e.g. PCI bus. That means device drivers need to be endianness-aware to work with both little-endian and big-endian processors.

AMD64 architecture is little-endian only. This causes more problems, because some applications are not properly tested on big-endian architectures.

Some examples of how endianness works in practice are below. They come from the hexdump of /bin/sh on amd64 and powerpc64.

0x7f454c46 is the magic number of ELF format (marked as yellow in the Figure 1) (0x7f followed by ELF in ASCII). In little-endian variant the values are reverted (offset 0x00). [5]

After the magic number and the number telling whether the architecture is 32-bit or 64-bit, follows the number which says whether the given architecture is little- or big-endian (1 meaning little-, 2 meaning big-, marked in blue).[6]

Then, in offset 0x20, there is a value which keeps the address of program header table. In 64-bit architectures, it is at offset 0x40 (marked in red).

```
0000000 457f 464c 0102 0901 0000 0000 0000 0000
0000010 0003 003e 0001 0000 f000 0000 0000 0000
0000020 0040 0000 0000 0000 11c0 0004 0000 0000
0000000 7f45 4c46 0202 0109 0000 0000 0000 0000
0000010 0002 0015 0000 0001 0000 0000 1003 6598
0000020 0000 0000 0000 0040 0000 0000 0002 aa28
```

Figure 1: Screenshot of hexdump /bin/sh from little-endian (upperside) and big-endian (downside)

FreeBSD gained support for POWER9 and the previous POWER8 in 12.0-RELEASE.[7]

3. Purpose of this work

My goal is to make powerpc64 platform equal to amd64 in terms of software support. This includes having usual desktop-class software like web browsers or desktop environments available. It also includes having server-class software like database servers.

4. Used hardware and methods

All tests were carried out using Talos Lite board with 4 core IBM POWER9 processor and 64GB RAM.

I used FreeBSD 12.0-RELEASE system with manually merged patches related to POWER from CURRENT branch and ran bulk builds of the whole ports tree using Poudriere.

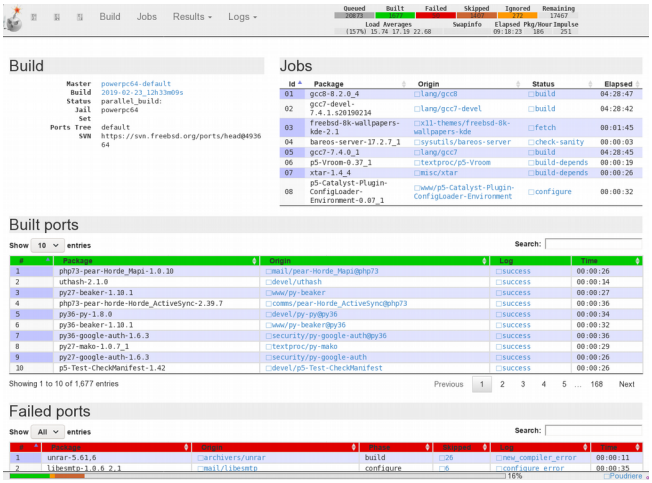

Figure 2: Screenshot of Poudriere web frontend

This screenshot shows the frontend of Poudriere accessible via web browser. I am able to easily see which ports fail to build and which ports are set not to build. It is also possible to browse specific compilation logs.

That allowed me to identify existing problems, fix them one by one, then launch next Poudriere run.

5. Results

Summary of identified problems

There are three main reasons why powerpc64 support in ports tree needs more work:

1. powerpc64 is big-endian on FreeBSD, while all Tier 1 architectures are little-endian.
2. powerpc64 uses (as do all other big-endian architectures on FreeBSD) outdated toolchain in the base system. The default compiler is GCC 4.2. Binutils 2.17 is also used.
3. FreeBSD uses powerpc64 name for 64-bit POWER port, but Linux uses ppc64 name.

1. Endianness problem is cultural and social. The most popular and widely available architectures, amd64 and arm, are little-endian. Developers often do not write software with endianness compatibility in mind.

   FreeBSD/powerpc64 little-endian port is in planning, but this is out-of-scope for this article. However, this will not solve all the problems, because:

   a) As a general rule, POWER processors older than POWER8 can only run big-endian (there are exceptions to that).

   b) Some people may prefer big-endian for various reasons.

   c) That would only solve powerpc64's problems, other big-endian architectures will be left with the same problem.

   d) Creating powerpc64le port will divide already small community of FreeBSD/powerpc* users to two groups – big-endian (powerpc, powerpc64 and powerpcspe) and little-endian (powerpc64le).

Endianness issues cause programs written for little-endian architectures to require byte-swapping functions. Common issues for little-endian-only software are mixed colors in graphic applications, because the hexadecimal value used for a given color is different when read in big-endian mode.

Example of such function is given below:

```
static inline int16_t
ORCT_Swapi16(int16_t x)
{
    return
    static_cast<uint16_t>((x << 8)
    | (x >> 8));
}
```

This code makes byte ordered in little-endian mode to be swapped to big-endian mode.

Unfortunately, the only solution to this problem is a political one – exposing big-endian architectures more and raise awareness that FreeBSD works on many different architectures. This is also troublesome for GNU/Linux ppc64 users and patches for many ports can be adapted to FreeBSD from GNU/Linux. There could also be a little-endian POWER architecture port of FreeBSD, but that would divide the (already small) FreeBSD community of POWER-users.

2. Old toolchain is a technical problem and is specific to FreeBSD.

When FreeBSD migrated from GNU toolchain to LLVM, it started with i386 and amd64 architectures. Soon after, arm platforms followed. This incidentally makes all little-endian architectures use LLVM.

However, all big-endian architectures are left with GCC 4.2 and Binutils 2.17.

The most common issues resulting from using outdated toolchain are:

a) plenty of software nowadays require C11 or C++11 compatibility. Because LLVM on all supported FreeBSD releases supports both, there are hundreds of example where the necessary `USES=compiler` is not put to given port's Makefile.

b) when the port defines `USES=compiler` properly, but its library dependency does not, this creates a linking issue. This happens because the port uses new GCC (and its new ABI), but the linked library is built with base GCC (and old GCC ABI). This problem requires adding `USES=compiler` to the library dependency, even though it compiles with base GCC. Example of such linking error:

```
/usr/local/lib/
libIlmThread.sodefined
reference t to
`std::__cxx11::basic_string
stream<char,
std::char_traits<char>,
std::allocator<char>
>::basic_stringstream(std::_
Ios_Openmode)@GLIBCXX_3.4.21
'
```

c) sometimes a LLVM-specific CXXFLAGS is put to Makefile. GCC in such situation can't compile such software and throws error.

The most prominent error is:

```
CXXFLAGS+=      -Wno-c++11-
narrowing
```

In above case, it is usually enough to instead force compilation in c++98 mode – LLVM compiles by default in c++11. This can be achieved by the following directive:

```
USE_CXXSTD=     c++98
```

d) a common error in software is redefining typedefs. This happens with code in example:

```
typedef struct _Example {
int a;
char b;
} Example;
```

```
typedef struct _Example
Example;
```

Code from the first fragment is put to a file that is included in another file, which has code from the former fragment.

In this case, it is enough to remove the typedef from the former fragment.

Alternatively, one could use appropriate USES directive to force GCC from ports to be used. New GCC allows typedefs to be redefined

The issue of outdated toolchain can be worked around by using newer GNU compiler from ports tree, but such workaround creates other issues:

e) programs do not respect `CXXFLAGS`, linking to base libstdc++, instead of libstdc++ from ports' GCC,

Example for devel/protobuf:

```
--- src/Makefile.am.bak
2018-10-27
21:56:16.784704000 +0200

+++ src/Makefile.am
2018-10-27
22:01:47.564751000 +0200

@@ -518,7 +518,7 @@

 # to build the js_embed
binary using $
```

```
(CXX_FOR_BUILD) so that it
is executable

 # on the build machine in a
cross-compilation setup.

 js_embed$(EXEEXT): $
(srcdir)/google/protobuf/com
piler/js/embed.cc

-       $(CXX_FOR_BUILD) -o
$@ $<

+       $(CXX_FOR_BUILD) $
{CXXFLAGS} -o $@ $<


js_well_known_types_sources
=                              \
google/protobuf/compiler/js/
well_known_types/any.js
\
google/protobuf/compiler/js/
well_known_types/struct.js
\
```

This specific issue alone made protobuf fail to build, resulting in over 400 ports to be skipped.

f) software developers believe that using FreeBSD implies having libc++ in base and add `-stdlib=libc++` to `CXXFLAGS`. This is not necessary, Clang uses libc++ by default and adding it breaks build with GCC.

g) GCC and LLVM include by default slightly different set of headers, resulting in some headers needed to be included manually when using GCC. The most common example is using `sys/types.h`, which contains commonly used typedefs (like uint).

3. There is also a third problem, which is that Linux uses ppc and ppc64 names for POWER architecture ports, while FreeBSD uses powerpc, powerpc64 and powerpcspe names. This issue is also present on amd64 and i386 architectures and is easily fixed:

```
ANT_ARCH=       $
{ARCH:S/amd64/x86-64/:S/i386
/x86/:S/powerpc64/ppc64/}
```

6. Conclusions

Switching to LLVM in base will allow focus only on architecture-related differences with amd64. This is already work-in-progress by some members of the community.[8] Having modern toolchain with C++17 support in FreeBSD base will fix most toolchain issues in ports.

With POWER9 available, there is a renewed interest in big-endian systems. Linux users port more and more software to big-endian architectures and FreeBSD/powerpc* will also be able to benefit from that.

7. References

1. Forever Mac, https://web.archive.org/web/20120930005749/http://www.forevermac.com/2005/10/apple-power-macintosh-g5-quad-core-2-5-ghz/, access from 22.02.2019,
2. Raptor Computing Systems, https://www.raptorcs.com/, access from 22.02.2019,
3. OpenPOWER Foundation, https://github.com/openbmc and https://github.com/open-power, access from 22.02.2019,
4. Mozilla, https://developer.mozilla.org/en-US/docs/Glossary/Endianness, access from 22.02.2019,
5. Unix System Laboratories, http://www.skyfree.org/linux/references/ELF_Format.pdf, page 1-5, access from 23.02.2019,
6. Unix System Laboratories, http://www.skyfree.org/linux/references/ELF_Format.pdf page 1-6 access from 23.02.2019,
7. The FreeBSD Documentation Project, https://www.freebsd.org/releases/12.0R/relnotes.html#hardware-support, access from 22.02.2019,
8. Alfredo Dal Ava, https://wiki.freebsd.org/powerpc/llvm-elfv2, access from 23.02.2019.