FreeBSD 13.0 Tool Chain

BY ED MASTE

FreeBSD 13.0 marks a significant milestone in the evolution of the FreeBSD tool chain: the completion of a decade-long migration to using a modern, permissively licensed compiler, linker, debugger, and miscellaneous binary utilities for all of FreeBSD's supported architectures.

From the beginning, FreeBSD relied on the GNU tool chain, including the GCC compiler, Binutils linker and binary utilities, and the GDB debugger. These tools were kept current with regular updates by a group of developers, both volunteer and paid. This continued until 2007 when the GNU project changed the license on these tools to version 3 of the GNU Public License (GPLv3), and they were not updated again.

A few years later, about twenty, developers met for a developer summit session at BSDCan 2010 to plan the future of the tool chain, with a goal of moving to a modern, permissively licensed tool chain. License considerations were only one part of the rationale. New technology, the ability to implement new and bespoke functionality, and promoting competition in OSS tools also factored into the team's goals.

At the same time, a number of tool-chain projects were firming up, both external to and within the FreeBSD project. Summit attendees noted that LLVM and Clang were maturing rapidly and gaining interest in the compiler research community. A BSD-licensed ELF tool-chain project was in progress, as well as a number of debugger projects.

From the beginning, FreeBSD relied on the GNU tool chain, including the GCC compiler, Binutils linker and binary utilities, and the GDB debugger.

The team set a goal of collaborating with these tool-chain projects and migrating to the new components. Individual components were added to FreeBSD as they became usable, often installed alongside the existing tool, but disabled by default. After testing and validation, they became default, sometimes on an architecture-by-architecture basis. Eventually, the new tool was enabled across all supported architectures and the old tool was removed. Aside from a small caveat with the debugger, this is now complete for all tool-chain components and all supported architectures. Read on for the full details.

Compiler

The compiler is one of the most important components in the tool chain and was the first major component that we migrated. The Clang compiler had its initial release in 2007 and was maturing quickly. Roman Divacky imported a version into FreeBSD's contrib software tree in

June 2010, and we provided it (as /usr/bin/clang) in FreeBSD 9.0 for only the x86 and powerpc architectures.

With substantial effort on the part of the Clang developers and integration effort from Roman, Ed Schouten, Dimitry Andric, and others, we enabled Clang as the default compiler (/usr/bin/cc) for i386 and amd64 in 2012. This shipped with FreeBSD 10.0.

In 2014, Clang was ready to be the default compiler for little-endian Arm. It was also provided on PowerPC, but not as the default compiler. This is the configuration shipped in FreeBSD 11.0. FreeBSD 11.0 also introduced 64-bit arm (AArch64) support, with Clang as the only available compiler.

Compiler updates continued in the lead up to FreeBSD 12.0, but architecture support remained the same. 2019 and 2020 saw significant upstream Clang/LLVM architecture improvements, and remaining architectures including RISC-V and MIPS switched to Clang by default. Sparc64 was the sole remaining architecture relying on the obsolete in-tree GCC version. When FreeBSD/sparc64 was retired, there was no need to keep GCC in the tree, and it was removed in early 2020.

Tool chains typically include an assembler. As of FreeBSD 13.0 we use Clang's integrated assembler (IAS) to assemble parts of the FreeBSD base system and no longer provide a standalone /usr/bin/as.

Using Clang as FreeBSD's standard compiler has made it a compelling system for full-system research. The University of Cambridge's Temporally Enhanced Security Logic Assertions (TESLA) and Capability Hardware Enhanced RISC Instructions (CHERI) research projects build on Clang/LLVM and benefit from the availability of a full, Clang-built operating system kernel and userland.

Compiler updates continued in the lead up to FreeBSD 12.0, but architecture support remained the same.

Linker

A linker combines object files and libraries into an executable or shared library. When the tool chain project started, no alternative linker with a clear path to viability existed. The MCLinker project demonstrated some early momentum, but ultimately did not support many features required by the FreeBSD build.

By 2015, LLVM's lld was making good progress, and we imported a snapshot in 2016. We started with it built by default but installed with a non-default filename. It was available via a compiler switch, -fuse-ld=lld.

By FreeBSD 12.1 lld was built for all architectures except sparc64 and riscv64 and was the default linker for 32- and 64-bit arm and x86. With the development of riscv64 support upstream in lld and sparc64's retirement, lld became the sole linker for all architectures in FreeBSD 13.0.

Binary Utilities

A tool chain includes a number of small tools for examining or processing object, library and executable files—ELF objects in the case of FreeBSD. This includes tools like size, strings, and readelf for examining files, objcopy for converting or processing objects, and ar for creating and extracting library archives. This category also includes libraries used by other tools to parse objects—for example, libelf and libdwarf.

In older FreeBSD versions, most of these tools were obtained from GNU binutils with a few

exceptions--for example, the archive manager was a bespoke tool based on libarchive. FreeBSD also implemented versions of libelf and libdwarf starting in 2006.

The ELF Tool Chain project began as a standalone effort in 2008, led by Joseph Koshy. The project imported some existing FreeBSD tools and libraries, and additional contributors joined the project. Kai Wang implemented many of the missing tools, including elfcopy/objcopy and readelf.

By 2015, the ELF Tool Chain versions of addr2line, elfcopy/objcopy, nm, size and strings were sufficiently functional that we switched to them by default and shipped them in FreeBSD 11.0.

This work did not address the assembler or linker; work began on both tools within ELF Tool Chain, but neither is functional at present. We addressed the assembler by relying on Clang's integrated assembler instead and switched to LLVM's lld for the linker for almost all architectures. Sparc64 was the sole architecture relying on the in-tree GNU ld. With the retirement of sparc64 support we were able to remove binutils from the FreeBSD tree prior to FreeBSD 13.0.

The migration from binutils also left FreeBSD without objdump in the base system. Much of the information provided by objdump is also available (in a slightly different format) from readelf, and LLVM provides an llvm-objdump that is largely compatible. It is not yet installed by default in the base system but will likely be enabled in a future version. Of course, GNU objdump is also available by installing the binutils port or package.

Source Level Debugger

The remaining tool chain component is the debugger, and LLVM provided a path forward here as well. LLDB is LLVM's debugger, described in detail in previous FreeBSD Journal articles. It builds on LLVM components for disassembly, uses Clang as the expression parser, and is scriptable via Python and Lua.

We added LLDB to the build as an experimental feature in 2013, prior to FreeBSD 10.0, and enabled it by default for amd64 and arm64 in 2015, for FreeBSD 11.0. In 2017, Karnajit Wangkhem submitted a patch adding support for i386 to the JIT expression engine, and we enabled it by default in FreeBSD 12.0.

LLDB includes FreeBSD target support for 32-bit Arm, as well as PowerPC and MIPS, although it is not well tested. Builds may be enabled by default after sufficient testing and integration. LLDB is now a functional FreeBSD userland debugger, but currently lacks kernel debugging support.

LLDB includes FreeBSD target support for 32-bit Arm, as well as PowerPC and MIPS.

With FreeBSD Foundation sponsorship, Moritz Systems recently fixed many outstanding bugs, improved arm64 target support, added an initial follow-fork mode, and is improving userland core file debugging. Projects to implement live kernel debugging and post-mortem kernel coredump support are under discussion. We also need to implement RISC-V target support.

CTF Tools

FreeBSD's DTrace support makes use of the Compact C Type Format (CTF), which provides the minimum required debug information to make C language types available to DTrace scripts. This currently uses three tools released under the Common Development and Distribution Li-

cense (CDDL). They implement CTF version 2 and are largely unchanged since the import of DTrace from OpenSolaris in 2008.

There are a number of alternative CTF tool implementations available. Contemporary binutils includes libctf, a library to parse and edit CTF data. The CTF format has also been extended to version 3 and more recently to version 4. Also, OpenBSD's Martin Pieuchot has implemented a minimal set of permissively licensed CTF tools. Future FreeBSD tool chain work will be needed to determine where we take the CTF tooling.

Diagnostic Tools

Valgrind is part of a set of tools for memory access and leak debugging and checking other aspects of program behavior. FreeBSD support for Valgrind has been maintained outside of the main Valgrind tree for almost two decades, and a patched Valgrind is available from the ports tree and packages collection. Many different developers have maintained and updated the FreeBSD Valgrind port over the years; most recently Paul Floyd has updated it to the latest released version 3.17.0 and is working to commit the FreeBSD changes upstream.

FreeBSD support for Valgrind has been maintained outside of the main Valgrind tree for almost two decades

Clang includes built-in support for various sanitizers that provide debugging and diagnostic information. AddressSanitizer detects memory errors such as out-of-bounds accesses or use-after-free and is available via the -fsanitize=address command line flag (using Clang in the base system). A related tool, MemorySanitizer, detects reads of uninitialized variables. ThreadSanitizer detects data races, and UndefinedBehaviourSanitizer catches undefined C behavior (such as signed integer overflow) at runtime. Additional sanitizers exist and will need more work to enable them on FreeBSD.

Clang also includes a static analyzer invoked via the scan-build front end. It is available with Clang installed from an LLVM package, not the base-system Clang. The static analyzer serves a purpose broadly similar to compiler warnings, but at a much higher level.

For code coverage analysis, LLVM provides llvm-cov. It is also installed as gcov and operates in a GNU, gcov-compatible mode when invoked via the alias. When a program is compiled with –coverage, it emits a .gcov file when it exits, and gcov uses that to display execution counts for each line (or basic block) of the source. Performance profiling is available via bespoke BSD licensed code: hwpmc kernel support and the pmcstat userland tool.

Future Work

With the transition to Clang/LLVM complete, the tool chain team is investigating enhancements we can build on this foundation. One of these is Link Time Optimization (LTO), intermodular optimization performed during the link stage. Essentially LTO uses object files and libraries containing LLVM intermediate representation (IR) instead of target binary ELF objects. This allows these to be combined at link time and have the LLVM optimization passes operate on the entire binary as a whole and not just individual compilation units. One caveat with LTO is that the nm and ar tools need to be able to parse LLVM IR symbol tables, and the ELF Tool Chain versions used by the base system do not have this capability. We will need to either extend ELF Tool Chain nm and ar or switch to the LLVM versions of these tools.

Another tool chain feature provided by Clang is Control Flow Integrity (CFI). CFI broadly refers to techniques used by the compiler to avoid run-time attempts to subvert a program's intended operation (control flow) by some malware that manages to gain execution. Clang's CFI support requires LTO and includes a number of individual checks to detect various cases of bad casts and invalid indirect calls.

The CHERI project presents another opportunity for future work in the FreeBSD tool chain. CheriBSD is a derivative of FreeBSD that implements CHERI memory protection and software compartmentalization and is maintained in an external repository. CHERI also includes an LLVM-based tool chain with support for a number of CHERI-enhanced ISAs.

ED MASTE manages project development for the FreeBSD Foundation. He is also a member of the elected FreeBSD Core Team. Aside from FreeBSD, he has contributed to a number of other open-source projects, including LLVM, ELF Tool Chain, QEMU, and Open vSwitch. He lives in Kitchener-Waterloo, Canada, with his wife, Anna, and children, Pieter and Daniel.