The History and Future of Core Dumps in FreeBSD

Sam W. Gwydir, Texas A&M University sam@samgwydir.com

December 29, 2016

Abstract

Crash dumps, also known as core dumps, have been a part of BSD since it's beginnings in UNIX. A core dump is "a copy of memory that is saved on secondary storage by the kernel" for debugging a system failure. Though 38 years have passed since =doadump= came about in Bell 32/V, core dumps are still needed and utilized in much the same way they were then. Given this one might assume the core dump code changed little over time but this assumption did not prove correct.

What has changed over time is where core dumps are sent to and what processor architectures are supported. Previous to UNIX, core dumps were printed to a line printer or punch cards. At the birth of UNIX core dumps were made to magnetic tape and because UNIX supported only the PDP-11, it was the only architecture supported for dumps. Over time the specific dump device has changed from tape, to hard disk and even over the network to a remote machine. In addition, machine architecture support has evolved from different PDP-11 models to hp300, i386 up to the present day with amd64 and ARM.

The following paper begins with a quick background on what core dumps are and why operators might need them. Following that a complete history of core dumps in UNIX and BSD is produced, the current state of the core dump facility and some of the more common extensions in use. We conclude with a call to action for modularizing the core dump code such that different methods of core dump can be dynamically loaded into the kernel on demand.

In addition a call to action will be made for modularizing the core dump code.

1 Introduction

The BSD core dump facility performs a simple yet vital service to the operator: preserving a copy of the contents of system memory at the time of a fatal error for later debugging.

Throughout the history of UNIX operating systems, different methods have been used to produce a core dump. The earliest (pre UNIX) core dumps were made directly to a line printer, taking some hours to print. In the earliest UNIXes magnetic tape was the only supported dump device but when hard disks support matured swap space was used, obviating the need for changing out tapes before a dump. Modern and embedded systems continue to introduce new constraints that have motivated the need for newer methods of ex-filtrating a core dump from a faltering kernel.

The FreeBSD variant of the BSD operating system has introduced gradual extensions to the core dumping facility. FreeBSD 6.2 introduced "minidumps", a subset of a full dump that only consists of active kernel memory . FreeBSD 7.1's `textdumps(4)` consist of the result of debugger scripting at the time of `panic(9)`. In FreeBSD 12 CURRENT public-key cryptographic encryption has introduced support for core dumps. Though not in the main source tree, compressed dumps and the ability to dump to a remote network device exist and function. While promising, these extensions have been inconsistent in their integration and interoperability. Notably, Mac OS X has also introduced similar compression and network dumping features into their kernel albeit with a distinct pedigree from FreeBSD.

The following paper will provide a historical survey of the dump facility itself, from its' introduction in UNIX to its' current form in modern BSDs and BSD derived operating systems. We will also explore these core dump extensions, associated tools,

and describe an active effort to fully modularize them, allowing the operator to enable one or more of them simultaneously.

2 Motivation

In UNIX and early BSD's core dumps were originally made to magnetic tape which was superseded by dumping to a swap partition on a hard disk since at least 3BSD. For decades since, increases in physical system memory and swap partition size have loosely tracked increases in available persistent memory, allowing for the continued use of this paradigm.

However, recent advances in commodity system hardware have upended the traditional memory to disk space ratio with systems now routinely utilizing 1TB or more physical memory whilst running on less than 256GB of solid state disk. Given that the kernel memory footprint has grown in size, the assumption that disk space would always allow for a swap partition large enough for a core dump has proved to be inaccurate. This change has spurred development of several extensions to the core dumping facility, including compressed dumping to swap and dumping over the network to a server with disk space for modern core dumps. Because dumps contain all the contents of memory any sensitive information in flight at the time of a crash appears in the dump. For this reason and others encrypted dumps have been recently added to FreeBSD.

In documenting current dump code it occurred to the author and his colleagues that the BSD dump code is not an unchanging relic, but a living breathing artifact that can be traced directly back to UNIX's birth. Hopefully the information herein is of use to inform further work on core dumps, failing that we hope it is interesting.

3 Background

3.1 Full Dump Contents

The canonical form of core dump is the "full dump". Full dumps are created via the `doadump()` code path which starts in `sys/kern/kern_shutdown.c`. The resulting dump is an ELF formatted binary written to a configured swap partition. The following is based on x86 and amd64 code and is the result of `dumpsys_generic()`. This will be similar in format but different values for different architectures.

Table 1: Full Dump Format

Field	Description
Leader	See Table 2
ELF Header	See Table 3
Program Headers	
Memory Chunks	
Trailer	See Table 2

Table 2: `kerneldumpheader` Format

Field	Value
magic	"FreeBSD Kernel Dump"
architecture	"amd64"
version	1 (kdh format version)
architectureversion	2
dumplength	varies, excluding headers
dumptime	current time
blocksize	block size
hostname	hostname
versionstring	version of OS
panicstring	message given to `panic(9)`
parity	parity bits

Table 3: ELF Header Format

Field	Value
`ehdr.e_ident[EI_MAG0]`	`0x7f`
`ehdr.e_ident[EI_MAG1]`	'E'
`ehdr.e_ident[EI_MAG2]`	'L'
`ehdr.e_ident[EI_MAG3]`	'F'
`ehdr.e_ident[EI_CLASS]`	2 (64-bit)
`ehdr.e_ident[EI_DATA]`	1 (little endian)
`ehdr.e_ident[EI_VERSION]`	1 (ELF version 1)
`ehdr.e_ident[EI_OSABI]`	255
`ehdr.e_type`	4 (core)
`ehdr.e_machine`	62 (x86-64)
`ehdr.e_phoff`	size of this header
`ehdr.e_flags`	`0`
`ehdr.e_ehsize`	size of this header
`ehdr.e_phentsize`	size of program header
`ehdr.e_shentsize`	size of section header

3.2 Modern Full Core Dump Procedure

When a UNIX-like system such as FreeBSD encounters an unrecoverable and unexpected error the kernel will "panic". Though the word panic has connotations of irrationality, the function `panic(9)` maintains composure while it "[terminates] the running system" and attempts to save a core dump to a configured dump device.

What follows is a thorough description of the FreeBSD core dump routine starting with `doadump()` in `sys/kern/kern_shutdown.c`.

`doadump()` is called by `kern_reboot()`, which shuts down "the system cleanly to prepare for reboot, halt, or power off." [4] `kern_reboot()` calls `doadump()` if the `RB_DUMP` flag is set and the system is not "cold" or already creating a core dump. `doadump()` takes a boolean informing it to whether or not to take a "text dump", a form of dump carried out if the online kernel debugger, `DDB`, is built into the running kernel. `doaddump()` returns an error code if the system is currently creating a dump, the dumper is NULL and returns error codes on behalf of `dumpsys()`.

`doadump(boolean_t textdump)` starts the core dump procedure by saving the current context with a call to `savectx()`. At this point if they are configured, a "text dump" can be carried out. Otherwise a core dump is invoked using `dumpsys()`, passing it a `struct dumper`. `dumpsys()` is defined on a per-architecture basis. This allows different architectures to setup their dump structure differently. `dumpsys()` calls `dumpsys_generic()` passing along the `struct dumperinfo` it was called with. `dumpsys_generic()` is defined in `sys/kern/kern_dump.c` and is the meat of the core dump procedure.

There are several main steps to the `dumpsys_generic()` procedure. The main steps are as follows. At any point if there is an error condition, goto failure cleanup at the end of the procedure.

1. Fill in the ELF header.
2. Calculate the dump size.
3. Determine if the dump device is large enough.
4. Begin Dump
   (a) Leader (Padding)
   (b) ELF Header
   (c) Program Headers
   (d) Memory Chunks
   (e) Trailer
5. End Dump

   After this is done the kernel writes out a NULL byte to "Signal completion, signoff and exit stage left." And our core dump is complete.

4 History

The following sections list when different features of the core dump code were introduced starting with the core dump code itself. First the dump facility will be followed through the different versions of Research UNIX and then BSD through to present versions of FreeBSD. A quick explanation of the state of Mac OS X's dump features will follow. Afterward the various core dump extensions' history will be explored.

4.1 Core Dumps in UNIX

Core dumping was initially a manual process. As documented in Version 6 AT&T UNIX's `crash(8)`, an operator could take a core dump "if [they felt] up to debugging". Though 6th Edition is not the first appearance of dump code in UNIX, it is the first complete repository of code the public has access to.

4.1.1 6th Edition UNIX

In 6th Edition UNIX `crash(8)` teaches us how to manually take a core dump:

> If the reason for the crash is not evident (see below for guidance on 'evident') you may want to try to dump the system if you feel up to debugging. At the moment a dump can be taken only on magtape. With a tape mounted and ready, stop the machine, load address 44, and start. This should write a copy of all of core on the tape with an EOF mark.

`/usr/sys/conf/m40.s` and `/usr/sys/conf/m45.s` give UNIX support for the PDP-11/40 and PDP-11-45.

4.1.2 7th Edition UNIX

4.1.3 Bell 32/V

Bell 32/V was an early port of UNIX to the DEC VAX architecture making use of the C programming language to decouple the code from the PDP-11. `/usr/src/sys/sys/locore.s` contains the first appearance of `doadump()` the same function name used today written in VAX assembly.

4.2 Core Dumps in BSD

4.2.1 1BSD & 2BSD

1BSD and 2BSD inherited their dump code directly from 6th Edition UNIX and therefore written for the PDP-11 supporting the PDP-11/40 and PDP-11/45.

4.2.2 3BSD

3BSD imports its' dump code from Bell 32/V maintaining the name `doadump`. Because of it's pedigree, `doadump` is written for in VAX assembly.

`usr/src/sys/sys/TODO` notes that "large core dumps are awful and even uninterruptible!".

4.2.3 4BSD

4BSD introduces a new feature to `doadump`, printing tracing information with `dumptrc`. In addition `usr/src/sys/sys/TODO` is the first mention of adding a dump to swap feature listing a "todo": "Support automatic dumps to paging area". before

4.2.4 4.1BSD

Beginning in 4.1BSD (actually showing up in 4.1c2BSD) `doadump` is relegated to setting up the machine for `dumpsys()` which is written in C and found in `sys/vax/vax/machdep.c`. `doadump` now fulfills the "todo" listed in 4BSD and dumps to the "paging area", or swap. `savecore(8)` is introduced to extract the core from the swap partition and place it in the filesystem.

4.2.5 4.2BSD

4.2.6 4.3BSD

Initial support is added for the "tahoe" processor.

4.2.7 4.3 BSD-Tahoe

Tahoe support is now mature and `sys/tahoe/tahoe/machdep.c` and `doadump` is ported to the tahoe. `savecore` is re-written in ANSI C. #`http://gunkies.org/wiki/4.3_BSD_Tahoe`

4.2.8 4.3 BSD Net/1

same as 4.3-Tahoe

4.2.9 4.3 BSD-Reno

`usr/src/sys/hp300/locore.s` and `usr/src/sys/i386/locore.s` introduce hp300 and i386 support respectively.

4.2.10 4.3 BSD Net/2

same as reno

4.2.11 4.4

Tahoe no longer compiles and therefore cannot dump. i386 does not compile. VAX does not compile. sparc is added Not yet done. `usr/src/sys/luna68k/luna68k/locore.s` introduces OMRON m68030 support including dump support.

4.2.12 4.4-BSD Lite1

same as 4.4 – changes made due to USL lawsuit.

4.2.13 4.4-BSD Lite2

-

4.2.14 BSD SCCS

-

4.2.15 386BSD-0.0

i386 support, hp300 support

4.2.16 386BSD-0.1

i386 support, hp300 support

4.2.17 386BSD-0.1-patchkit

i386 support, hp300 support

4.2.18 FreeBSD 1.0, 1.1, 1.1.5

i386 support, hp300 support from 386BSD-0.1-patchkit

4.2.19 FreeBSD 2.0 2.0.5, 2.1.0, 2.1.5, 2.1.6, 2.1.6.1, 2.1.7, 2.2.0, 2.2.1, 2.2.2, 2.2.5, 2.2.6, 2.2.7, 2.2.8

Pulls in 4.4BSD-Lite1 code for hp300, luna68k, news3400, pmax, sparc, tahoe and vax

4.2.20 FreeBSD 3.0.0, 3.1.0, 3.2.0, 3.3.0, 3.4.0, 3.5.0

4.2.21 FreeBSD 4.0.0 4.1.0, 4.1.1, 4.2.0, 4.3.0, 4.4.0, 4.5.0, 4.6.0, 4.6.1, 4.6.2, 4.7.0, 4.8.0, 4.9.0, 4.10.0, 4.11.0

4.2.22 FreeBSD 5.0.0 5.1.0, 5.2.0, 5.2.1, 5.3.0, 5.4.0, 5.5.0

4.2.23 FreeBSD 6.0.0, 6.1.0, 6.2.0, 6.3.0, 6.4.0

- Write on minidump

4.2.24 FreeBSD 7.0.0, 7.1.0, 7.2.0, 7.3.0, 7.4.0

- Write on textdump

4.2.25 FreeBSD 8.0.0, 8.1.0, 8.2.0, 8.3.0, 8.4.0

4.2.26 FreeBSD 9.0.0, 9.1.0, 9.2.0

4.2.27 FreeBSD 10.0.0, 10.1.0, 10.2.0, 10.3.0

4.2.28 FreeBSD 11.0.0, 11.0.1

4.2.29 FreeBSD 12-CURRENT

- Write on encrypted textdump

  commit f63c437216e0309e4a319c2c95a2f8ca061c0bca Author: def <def@FreeBSD.org> Date: Sat Dec 10 16:20:39 2016 +0000

  Add support for encrypted kernel crash dumps.

Not yet done.

4.3 Core Dumps in Mac OS X

Mac OS X is capable of creating gzipped core dumps and dumping locally, or over the network using a modified `tftpd(8)` daemon they call `kdumpd(8)`. In addition dumps over FireWire are supported for situations where the kernel panic is caused by the Ethernet driver of network code.

In `xnu/osfmk/kdp/kdp_core.c` Mac OS X gzips its' core dump before writing it out to disk, and is otherwise much like the FreeBSD "full dump" procedure with one major difference. Notably, Mac OS X uses a different executable image-format called Mach-O, as opposed to ELF, because OS X runs a hybrid Mach and BSD kernel called XNU.

Network dumping "has been present since Mac OS X 10.3 for PowerPC-based Macintosh systems, and since Mac OS X 10.4.7 for Intel-based Macintosh systems." From `kdumpd(8)`:

> Kdumpd is a server which receives kernel states in the form of a core dump from a remote Mac OS X machine.
>
> ...
>
> The kdumpd command is based on Berkeley tftpd(8) by way of FreeBSD, with several modifications.

4.4 BSD Core Dump Extensions

Not yet done.

4.4.1 `netdump` - Network Dump

- Netdumps
  - Duke University code from long ago
  - Picked up by Ed Maste at Sandvine, dropped
  - Picked up by Mark Johnston at Sandvine
  - Maintained by Mark Johnston at Isilon

Not yet done.

4.4.2 Compressed Dump

- Maintained by Mark Johnston at Isilon

Not yet done.

4.4.3 `minidumpsz` - Minidump Size Estimation

Not yet done. Created by Rodney W. Grimes for the author's work at Groupon. `minidumpsz` is a kernel module that can do an online estimation of the size of a minidump if it were to occur at the time `sysctl debug.mini_dump_size` is called.

5 Conclusion

Not yet done.

6 Acknowledgments

The author would like thank Michael Dexter for his help debugging the original issues that led to our current combined knowledge of core dumps. In addition Rodney W. Grimes' help reading code, from PDP-11 assembly to modern C, along with his historical knowledge was invaluable.

The author thanks Deb Goodkin of the FreeBSD Foundation for her help bringing the author into the FreeBSD community and lastly thanks the FreeBSD community in general for making this day and paper possible.

7 References

- [1] The Design and Implementation of the FreeBSD operating system by McKusick, Neville-Neil, and Watson
- [2] crash(8) - 3BSD
- [3] man 9 panic - `https://www.freebsd.org/cgi/man.cgi?query=panic&apropos=0&sektion=0&manpath=FreeBSD+10.3-RELEASE+and+Ports&arch=default&format=html`
- [4] `kern_shutdown.c` - `sys/kern/kern_shutdown.c`
- [5] Unix History Repository - `https://github.com/dspinellis/unix-history-repo`
- [6] A Repository with 44 Years of Unix Evolution - `http://www.dmst.aueb.gr/dds/pubs/conf/2015-MSR-Unix-History/html/Spi15c.html`
- `https://en.wikipedia.org/wiki/Core_dump`