The History and Future of Core Dumps in FreeBSD

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

December 24, 2016

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 exfiltrating 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 similiar 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 related 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In UNIX and early BSD’s core dumps were originally made to magnetic tape which was superseceded by dumping to a swap partition on a hard disk since at least 3BSD.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 hist 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 furthur work on core dumps, failing that we hope it is interesting.

3 Background

3.1 Core Dump Contents

3.2 Modern 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. [2] Core dumps, sometimes known as crash dumps are used to debug system failures. Crash dumps are "a copy of memory that is saved on secondary storage by the kernel when a catastrophic failure occurs." [1] What follows is a thorough description of the FreeBSD core dump routine starting with `doadump()` in `sys/kern/kern_shutdown.c`. The FreeBSD operating system source code can be checked out using subversion by following the FreeBSD handbook instructions [3].

`doadump()` is called by `kern_reboot()`, which shutsdown "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()` and then invokes a core dump using `dumpsys()`, passing it a `struct dumper` and optionally a "text dump" which is carried out if the online kernel debugger, `DDB`, is built into the running kernel.

`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 dumping 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

4 History

4.1 Core Dumps in UNIX

4.1.1 5th Edition UNIX

`usr/sys/conf/mch.s`

4.1.2 6th Edition UNIX

`/usr/sys/conf/m40.s`

4.1.3 7th Edition UNIX

4.1.4 Bell 32/V

4.2 Core Dumps in BSD

4.2.1 1BSD & 2BSD

- Uses v6 dump code

4.2.2 3BSD

4.2.3 4BSD

- add trace information with _dumptrc
- First talk of dump to swap in `/usr/src/sys/sys/TODO`

4.2.4 4.1c2BSD

- doadump calls dumpsys and is all in C now

4.2.5 4.1BSD

- Back to asm? Actually I might be wrong, it might be a C/asm hybrid right now

4.2.6 4.2BSD

The following is a quick history of core dumps in the BSD operating systems tracing from before the advent of `doadump` in 3BSD through to the present and a discussion of current work on compressed dumps, dumping over the network and encrypted dumps.

Core dumping was initially a manual process as documented in Version 6 AT&T UNIX's `crash(8)`, an operator, "if [they felt] up to debugging" would:

> With a tape mounted and ready, stop the machine, load address 44, and start.

Providing the operator with a core dump on tape to debug a crashed system.

As of 3BSD and with the advent of the LSI-11 core dumping has been automated via `doadump` ^[2], the same function name used today. `doadump` was added to 3BSD in 1980 by Ozalp Babaoglu and was written in 33 lines of PDP-11 assembly.

Beginning in 4.1BSD `doadump` was rewritten in C for the VAX and placed in `sys/vax/vax/machdep.c`.

4.3 Core Dumps in Mac OS X

`osfmk/kdp/kdp_core.c`

- gzipped
- net dump using kdumpd

4.4 BSD Core Dump Extensions

4.4.1 `netdump` - Network Dump

4.4.2 Compresssed Dump

4.4.3 Encrypted Dump

4.4.4 `minidumpsz` - Minidump Size Estimation

5 Conclusion

6 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