The History and Future of Core Dumps in FreeBSD

Sam Gwydir, Texas A&M University

December 23, 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, or `panic(9)`. This copy can represent a machine readable dump of the contents of system memory, a subset of those kernel pages that are active at the time of the crash, or a less complete but human readable form using debugger scripting.

The FreeBSD variant of the BSD operating system has introduced gradual extensions to the core dumping facility in the form of, "minidumps" that only represent active kernel memory, "textdumps(4)" consisting of the result of debugger scripting at the time of panic, encrypted dumps, compressed dumps and the ability to dump to a remote network device. While promising, these extensions have been inconsistent in their integration and interoperability.

This paper will provide a historical survey of the dump facility itself, these dump extensions, and describe an active effort to fully modularize them, allowing the operator to enable one or more of them simultaneously. It will also address related utilities to determine the size of a dump in advance and kernel debugger (DDB) scripting options.

2 Motivation

Though core dumps were originally made to magnetic tape, dumps have been made to a swap partition on a hard disk since at least 3BSD.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 relatively small solid state disks. 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 and dumping over the network to a server with disk space for modern core dumps. Network Dumping, or `netdump` does have some security implications which recent work on encrypted dumping may resolve.

3 Background

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 Research UNIX Version 5

`usr/sys/conf/mch.s`

```
.globl dump
dump:
  mov $4,r0 / overwrites trap vectors
  mov r1,(r0)+
  mov r2,(r0)+
  mov r3,(r0)+
  mov r4,(r0)+
  mov r5,(r0)+
  mov sp,(r0)+
  mov $KISA0,r1
  mov $8.,r2
1:
  mov (r1)+,(r0)+
  sob r2,1b
  mov $MTC,r0
  mov $60004,(r0)+
  clr 2(r0)
1:
  mov $-512.,(r0)
  inc -(r0)
2:
  tstb (r0)
  bge 2b
  tst (r0)+
  bge 1b
  5
  mov $60007,-(r0)
  br .
```

4.1.2 Research UNIX Version 6

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

```
.globl dump
dump:
  bit $1,SSR0
  bne dump

/ save regs r0,r1,r2,r3,r4,r5,r6,KIA6
/ starting at abs location 4
```

```
  mov r0,4
  mov $6,r0
  mov r1,(r0)+
  mov r2,(r0)+
  mov r3,(r0)+
  mov r4,(r0)+
  mov r5,(r0)+
  mov sp,(r0)+
  mov KISA6,(r0)+

/ dump all of core (ie to first mt error)
/ onto mag tape. (9 track or 7 track 'binary')

  mov $MTC,r0
  mov $60004,(r0)+
  clr 2(r0)
1:
  mov $-512.,(r0)
  inc -(r0)
2:
  tstb (r0)
  bge 2b
  tst (r0)+
  bge 1b
  reset

/ end of file and loop

  mov $60007,-(r0)
  br .
```

/usr/sys/conf/m45.s

```
/ Mag tape dump
/ save registers in low core and
/ write all core onto mag tape.
/ entry is thru 44 abs

.data
.globl dump
dump:
  bit $1,SSR0
  bne dump

/ save regs r0,r1,r2,r3,r4,r5,r6,KIA6
/ starting at abs location 4

  mov r0,4
  mov $6,r0
  mov r1,(r0)+
  mov r2,(r0)+
  mov r3,(r0)+
  mov r4,(r0)+
  mov r5,(r0)+
  mov sp,(r0)+
  mov KDSA6,(r0)+

/ dump all of core (ie to first mt error)
/ onto mag tape. (9 track or 7 track 'binary')

  mov $MTC,r0
  mov $60004,(r0)+
clr 2(r0)
1:
  mov $-512.,(r0)
  inc -(r0)
2:
  tstb (r0)
  bge 2b
  tst (r0)+
  bge 1b
  reset

/ end of file and loop

  mov $60007,-(r0)
  br .
```

4.1.3 Research UNIX v7

/usr/sys/conf/mch.s

```
/ Mag tape dump
/ save registers in low core and
/ write all core onto mag tape.
/ entry is thru 44 abs

.data
.globl dump
dump:

/ save regs r0,r1,r2,r3,r4,r5,r6,KIA6
/ starting at abs location 4

  mov r0,4
  mov $6,r0
  mov r1,(r0)+
  mov r2,(r0)+
  mov r3,(r0)+
  mov r4,(r0)+
  mov r5,(r0)+
```

```
  mov sp,(r0)+
  mov KDSA6,(r0)+

/ dump all of core (ie to first mt error)
/ onto mag tape. (9 track or 7 track 'binary')

.if HTDUMP
  mov $HTCS1,r0
  mov $40,*$HTCS2
  mov $2300,*$HTTC
  clr *$HTBA
  mov $1,(r0)
1:
  mov $-512.,*$HTFC
  mov $-256.,*$HTWC
  movb $61,(r0)
2:
  tstb (r0)
  bge 2b
  bit $1,(r0)
  bne 2b
  bit $40000,(r0)
  beq 1b
  mov $27,(r0)
.endif
HT = 0172440
HTCS1 = HT+0
HTWC = HT+2
HTBA = HT+4
HTFC = HT+6
HTCS2 = HT+10
HTTC = HT+32

MTC = 172522
.if TUDUMP
  mov $MTC,r0
  mov $60004,(r0)+
  clr 2(r0)
1:
  mov $-512.,(r0)
  inc -(r0)
2:
  tstb (r0)
  bge 2b
  tst (r0)+
  bge 1b
  reset

/ end of file and loop

  mov $60007,-(r0)
.endif
  br .
```

4.1.4 Bell 32/V

/usr/src/sys/sys/locore.s

```
#  0x200
# Produce a core image dump on mag tape
  .globl doadump
doadump:
  movl sp,dumpstack # save stack pointer
  movab dumpstack,sp # reinit stack
  mfpr $PCBB,-(sp) # save u-area pointer
  mfpr $MAPEN,-(sp) # save value
  mfpr $IPL,-(sp) # ...
  mtpr $0,$MAPEN # turn off memory mapping
  mtpr $HIGH,$IPL # disable interrupts
  pushr $0x3fff # save regs 0 - 13
  calls $0,_dump # produce dump
  halt

  .data
  .align 2
  .globl dumpstack
  .space 58*4 # seperate stack for tape dumps
dumpstack:
  .space 4
  .text
```

4.2 Core Dumps in BSD

4.2.1 1BSD & 2BSD

- Uses v6 dump code

4.2.2 3BSD

/usr/src/sys/sys/locore.s doadump

```
# ====================================
# Produce a core image dump on mag tape
# ====================================
  .globl doadump
doadump:
  movl sp,dumpstack # save stack pointer
  movab dumpstack,sp # reinit stack
  mfpr $PCBB,-(sp) # save u-area pointer
```

```
  mfpr $MAPEN,-(sp) # save value
  mfpr $IPL,-(sp) # ...
  mtpr $0,$MAPEN # turn off memory mapping
  mtpr $HIGH,$IPL # disable interrupts
  pushr $0x3fff # save regs 0 - 13
  calls $0,_dump # produce dump
  halt

  .data
  .align 2
  .globl dumpstack
  .space 58*4 # separate stack for tape dumps
dumpstack:
  .space 4
  .text
```

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 `netdump`

4.3.1 OS X Kernel Dump

`osfmk/kdp/kdp_core.c`

- gzipped
- net dump using kdumpd

4.4 Compresssed Dump

4.5 Encrypted Dump

5 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`