Efficient and Practical Capturing of Crash Data on Embedded Systems

John Ogness

Linutronix GmbH

2023-06-30

What are core dumps?

```
$ man 5 core

core(5)                    File Formats Manual                    core(5)

NAME
       core - core dump file

DESCRIPTION
       The  default action of certain signals is to cause a process
       to terminate and produce a core dump file, a file containing
       an image of the process's memory at the time of termination.
       This image can be used  in  a  debugger  (e.g.,  gdb(1))  to
       inspect  the  state  of  the  program  at  the  time that it
       terminated.  A list of the signals which cause a process  to
       dump core can be found in signal(7).
```

Core files utilize the ELF file format to organize the various elements of the process image.

background

Core Dumps

advantages

- functionality provided by the kernel
- all process data available (registers, stacks, heap, ...)
- post-mortem debugging
- offline debugging

background

Core Dumps

advantages

- functionality provided by the kernel
- all process data available (registers, stacks, heap, ...)
- post-mortem debugging
- offline debugging

disadvantages

- large storage requirements
- debugging tools required for analysis
- no information about other processes

The minicoredumper Project

Primary Goals

- minimal core dumps
- custom core dumps
- state snapshots

overview

EMBEDDED OPEN SOURCE SUMMIT

The minicoredumper Project

Primary Goals

- minimal core dumps
- custom core dumps
- state snapshots

Main Components

- minicoredumper
- libminicoredumper
- live dumps

minicoredumper

What is the minicoredumper?

- userspace application to extend the Linux core dump facility
- configuration files to specify desired data
- per-application configuration files
- in-memory compression features
- few dependencies
- no kernel patches required

minicoredumper

How is this possible from userspace?

```
$ man 5 core
[...]

   Naming of core dump files
       By default, a core dump file is named core, but the
       /proc/sys/kernel/core_pattern file (since Linux 2.6 and
       2.4.21) can be set to define a template that is used to name
       core dump files. The template can contain % specifiers
       which are substituted by the following values when a core
       file is created:
[...]

   Piping core dumps to a program
       Since Linux 2.6.19, Linux supports an alternate syntax for
       the /proc/sys/kernel/core_pattern file. If the first
       character of this file is a pipe symbol (|), then the
       remainder of the line is interpreted as the command-line for
       a user-space program (or script) that is to be executed.
```

minicoredumper

/proc/sys/kernel/core_pattern

Inform the kernel to use the minicoredumper for core dumps.

```
$ echo '|/usr/sbin/minicoredumper %P %u %g %s %t %h %e' \
                  | sudo tee /proc/sys/kernel/core_pattern
$ echo 0x7fffffff | sudo tee /proc/sys/kernel/core_pipe_limit

$ man 5 core
[...]
           %P   PID  of  dumped  process, as seen in the initial PID
                namespace (since Linux 3.12).
           %u   Numeric real UID of dumped process.
           %g   Numeric real GID of dumped process.
           %s   Number of signal causing dump.
           %t   Time of dump, expressed as seconds since the  Epoch,
                1970-01-01 00:00:00 +0000 (UTC).
           %h   Hostname (same as nodename returned by uname(2)).
           %e   The process or thread's comm value, which  typically
                is the same as the executable filename (without path
                prefix,  and  truncated  to  a  maximum  of  15
                characters),  but  may  have  been  modified  to  be
                something  different;  see  the  discussion  of
                /proc/pid/comm   and   /proc/pid/task/tid/comm   in
                proc(5).
```

minicoredumper

Configuration

configuration file

- JSON format
- specifies dump path
- specifies matching rules for ”recepts” (application-specific dump configurations)

minicoredumper

Configuration

configuration file

- JSON format
- specifies dump path
- specifies matching rules for ”recepts” (application-specific dump configurations)

recept file

- JSON format
- general features (stacks, threads, ...)
- specific memory mappings
- specific symbols
- compression options

minicoredumper

minicoredumper.cfg.json

Configuration file example:

```
{
    "base_dir": "/var/crash/minicoredumper",
    "watch": [
        {
            "exe": "*/realpath_example_app",
            "recept": "/etc/minicoredumper/example.recept.json"
        },
        {
            "comm": "example_app",
            "recept": "/etc/minicoredumper/example.recept.json"
        },
        {
            "exe": "/usr/bin/*"
        },
        {
            "recept": "/etc/minicoredumper/generic.recept.json"
        }
    ]
}
```

minicoredumper

example.recept.json

```
{
    "stacks": {
        "dump_stacks": true,
        "first_thread_only": true,
        "max_stack_size": 16384
    },
    "maps": {
        "dump_by_name": [
            "[vdso]"
        ]
    },
    "buffers": [
        {
            "symname": "my_allocated_struct_ptr",
            "follow_ptr": true,
            "data_len": 42
        }
    ],
    "compression": {
        "compressor": "xz",
        "extension": "xz",
        "in_tar": true
    },
    "write_proc_info": true
}
```

minicoredumper

How It Works

identify process data

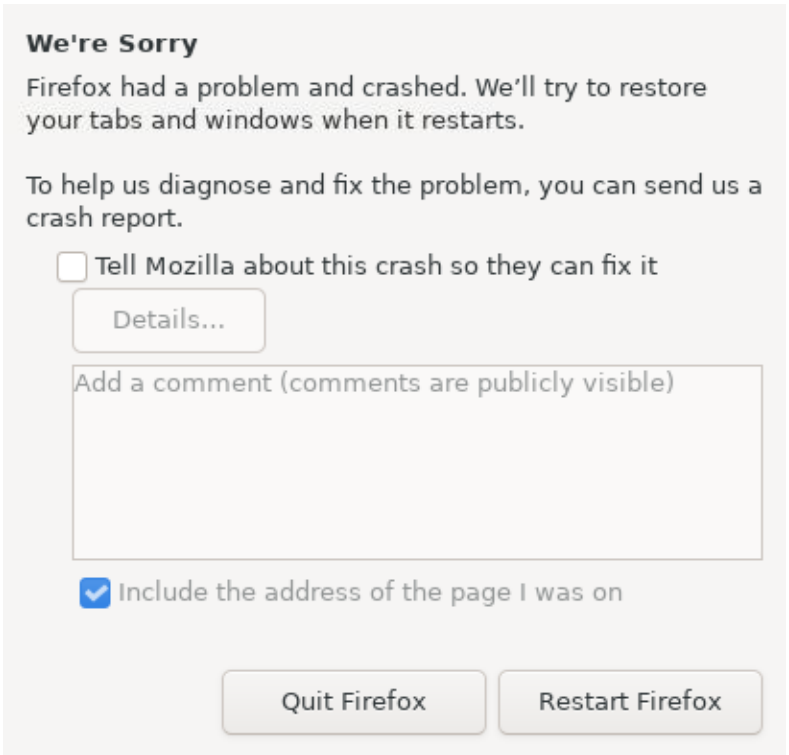
- ELF header from `stdin` (virtual memory allocations, symbols, shared objects, relocation, debug objects, ...)
- `/proc/N/maps` (memory maps)
- `/proc/N/stat` (stack pointers)
- `/proc/N/auxv` (auxiliary vector)
- `/proc/N/mem` (memory access)

dump process data

- write core as sparse file
- append custom ELF section note
- in-memory compression (with tar format support)

Simulate Core Dump

```
$ kill -s SEGV `pidof firefox-esr`
```



minicoredumper

Core Size Comparisons

default = default Linux core dump facility settings
minicore/* = default minicoredumper settings
minicore/1 = minicore/* changed to only first thread

type	file size	disk usage	core.tar.xz
default	448,820 KB	170,788 KB	17,676 KB
minicore/*	447,930 KB	2,328 KB	108 KB
minicore/1	446,630 KB	1,364 KB	72 KB

The full backtrace of the crashed thread is available in all variations.

Custom ELF Section Note

The custom ELF section note contains a list of ranges within the core file that are valid dump data.

```
$ eu-readelf -a core
[...]

Section Headers:
[Nr] Name                              Type     Addr     Off      Size
[ 0]                                   NULL     00000000 00000000 00000000
[ 1] .shstrtab                         STRTAB   00000000 1b56e6fc 00000030
[ 2] .debug                            PROGBITS 00000000 000183e0 1b554c20
[ 3] .note.minicoredumper.dumplist     NOTE     00000000 1b56d000 000016fc

[...]

Note section [ 3] '.note.minicoredumper.dumplist' of 5884 bytes
at offset 0x1b56d000:
  Owner          Data size  Type
  minicoredumper 5856       <unknown>: 80
```

Dependencies

With few dependencies, the minicoredumper can be added to existing systems with a relatively low storage cost.

```
$ LD_TRACE_LOADED_OBJECTS=1 /usr/sbin/minicoredumper \
                            | grep = | cut -d ' ' -f 1
libelf.so.1
libjson-c.so.5
libthread_db.so.1
libc.so.6
libz.so.1
```

Summary

The minicoredumper application itself is a very useful tool for providing powerful post-mortem debugging capabilities for an embedded system.

- **low storage overhead**
- **no runtime overhead**
- **simple configuration**
- **useful crash data**
- **very small dumps (even most EEPROM's would suffice!)**

Summary

The minicoredumper application itself is a very useful tool for providing powerful post-mortem debugging capabilities for an embedded system.

- **low storage overhead**
- **no runtime overhead**
- **simple configuration**
- **useful crash data**
- **very small dumps (even most EEPROM's would suffice!)**

But wait! There's more...

libminicoredumper

What is libminicoredumper?

- userspace library that allows applications to register specific data for dumping
- data can be dumped in-core and/or in external files
- data can be text-formatted and placed in external files
- data can be unregistered for dumping during runtime
- few dependencies

libminicoredumper

What is libminicoredumper?

- userspace library that allows applications to register specific data for dumping
- data can be dumped in-core and/or in external files
- data can be text-formatted and placed in external files
- data can be unregistered for dumping during runtime
- few dependencies

Why is this interesting?

- minimize dumped application data
- dump internal application data
- external dump files (text and binary) can provide insight into the problem without the need of a debugger

libminicoredumper

How It Works

- libminicoredumper exports two special symbols
  - `mcd_dump_data_version` (data format version number)
  - `mcd_dump_data_head` (linked list of dump registrations)
- when an application crashes, the minicoredumper looks for these symbols
- if the symbols are found, the minicoredumper can identify what and how the extra registered data is to be dumped

```
$ objdump -T /usr/lib/libminicoredumper.so.2.0.1 \
          | grep '\sDO\s'
00004098 g DO .data 00000004 Base mcd_dump_data_version
000040c0 g DO .bss  00000008 Base mcd_dump_data_head
```

libminicoredumper

API

```
int mcd_dump_data_register_bin(const char *ident,
                               unsigned long dump_scope,
                               mcd_dump_data_t *save_ptr,
                               void *data_ptr, size_t data_size,
                               enum mcd_dump_data_flags flags);

int mcd_dump_data_register_text(const char *ident,
                                unsigned long dump_scope,
                                mcd_dump_data_t *save_ptr,
                                const char *fmt, ...);

int mcd_vdump_data_register_text(const char *ident,
                                 unsigned long dump_scope,
                                 mcd_dump_data_t *save_ptr,
                                 const char *fmt, va_list ap);

int mcd_dump_data_unregister(mcd_dump_data_t dd);
```

libminicoredumper

Example Application (mycrasher)

```
int main(void)
{
    mcd_dump_data_t d[3];
    char *x = NULL;
    char *s;
    int *i;

    s = strdup("my string");
    i = malloc(sizeof(*i));
    *i = 42;

    mcd_dump_data_register_bin(NULL, 1024, &d[0], s, strlen(s) + 1,
                               MCD_DATA_PTR_DIRECT | MCD_LENGTH_DIRECT);

    mcd_dump_data_register_bin("i.bin", 1024, &d[1], i, sizeof(*i),
                               MCD_DATA_PTR_DIRECT | MCD_LENGTH_DIRECT);

    mcd_dump_data_register_text("out.txt", 1024, &d[2],
                                "s=\"%s\" *i=%d\n", s, i);

    *x = 0; /* BOOM! */
```

libminicoredumper

Example Application Debugging

```
$ ./mycrasher
Segmentation fault (core dumped)

$ sudo mv /.../mycrasher.20230624.184537+0200.42669 .

$ sudo chown -R `id -u` mycrasher.20230624.184537+0200.42669

$ cd mycrasher.20230624.184537+0200.42669

$ find . -type f | sort
./core.tar.xz
./dumps/42669/i.bin
./dumps/42669/out.txt
./symbol.map
```

The `symbol.map` file contains the core file information for all the external binary dumps.

```
$ cat dumps/42669/out.txt
s="my string" *i=42
```

Example Application Debugging (cont)

```
$ tar -xJSf core.tar.xz

$ gdb ../mycrasher core
[...]
Core was generated by `./mycrasher'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000055b908f00259 in main () at mycrasher.c:25
25          *x = 0; /* BOOM! */
(gdb) print s
$1 = 0x55b90a60b2a0 "my string"
(gdb) print i
$2 = (int *) 0x55b90a60b2c0
(gdb) print *i
$3 = 0
```

Unlike for `s`, the data pointed to by `i` is not available in the core file because it was stored externally in `i.bin`.

Example Application Debugging (cont)

Using the coreinject tool, external binary dumps can be inserted into the core files.

```
$ coreinject core symbol.map dumps/42669/i.bin
injected: i.bin, 4 bytes, direct

$ gdb ../mycrasher core
[...]
Core was generated by `./mycrasher'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000055b908f00259 in main () at mycrasher.c:25
25          *x = 0; /* BOOM! */
(gdb) print s
$1 = 0x55b90a60b2a0 "my string"
(gdb) print i
$2 = (int *) 0x55b90a60b2c0
(gdb) print *i
$3 = 42
```

libminicoredumper

Dependencies

With few dependencies, the libminicoredumper can be added to custom applications with a relatively low storage cost.

```
$ objdump -x /usr/lib/libminicoredumper.so.2.0.1 \
          | grep NEEDED
  NEEDED               libc.so.6
```

libminicoredumper

Summary

The libminicoredumper allows applications to provide very fine-tuned data dumps at a minimal cost.

- **low storage overhead**
- **no runtime overhead, but be aware registration/unregistration invokes memory allocations, locking, list searching**
- **simple API**
- **precise data specification**
- **runtime dump registration changes supported**

libminicoredumper

Summary

The libminicoredumper allows applications to provide very fine-tuned data dumps at a minimal cost.

- **low storage overhead**
- **no runtime overhead, but be aware registration/unregistration invokes memory allocations, locking, list searching**
- **simple API**
- **precise data specification**
- **runtime dump registration changes supported**

But wait! There's more...

live dumps

EMBEDDED OPEN SOURCE SUMMIT

What are live dumps?

- dump registered data for running applications
- dumps can be triggered on crash
- dumps can be triggered manually
- few dependencies

live dumps

EMBEDDED OPEN SOURCE SUMMIT

What are live dumps?

- dump registered data for running applications
- dumps can be triggered on crash
- dumps can be triggered manually
- few dependencies

Why is this interesting?

- allows pseudo state snapshots

live dumps

How It Works

minicoredumper_regd

- creates UNIX local domain datagram socket with abstract address
- socket receives credentials to identify sender PID
- maintains a list of PID's in shared memory of applications with registered dumps

```
$ ss -l | grep minicoredumper
u_dgr UNCONN 0 0 @minicoredumper.42850 262696 * 0
u_dgr UNCONN 0 0 @minicoredumper       262695 * 0

$ ls -l /dev/shm/minicoredumper.shm
-rw------- 1 mcd mcd 56 Jun 24 19:09 /dev/shm/minicoredumper.shm
```

How It Works (cont)

libminicoredumper

- registers itself with minicoredumper_regd via UNIX local domain socket on first data dump registration
- unregisters itself from minicoredumper_regd via UNIX local domain socket on last data dump unregistration

live dumps

How It Works (cont)

minicoredumper (an application crashed)

- read PID list from shared memory
- for each thread associated with each PID, attach and freeze the task using `PTRACE_SEIZE` and `PTRACE_INTERRUPT`, respectively
- for each PID, dump the registered data (via `/proc/N/mem`)
- for each thread associated with each PID, detach from the task using `PTRACE_DETACH`
- perform the dumps for the crashing application

Dependencies

With few dependencies, the minicoredumper_regd can be added to existing systems with a relatively low storage cost.

```
$ LD_TRACE_LOADED_OBJECTS=1 /usr/sbin/minicoredumper_regd \
                            | grep = | cut -d ' ' -f 1
libc.so.6
```

live dumps

Pseudo State Snapshots

- latencies between dumps vary greatly depending on hardware, system load, application, number of registered applications, ...
- expect latencies from 2ms to 30ms between crash event and the first dump
- expect latencies from 30us to 4ms between all successive dumps

live dumps

Summary

Live dumps can be useful for capturing a pseudo state snapshot of various related applications if any one should crash or by manually triggering it using the minicoredumper_trigger tool.

- low storage overhead
- dumps data for multiple applications, **but** be aware of latencies between dumps
- no runtime overhead, **but** be aware of application freezing during dumps

Project Status

- current release version 2.0.6 (presented here)
- packages available for Debian, OpenEmbedded, gentoo
- proof-of-concept for gdb dump list support
- TODO: implement modern tar format
- TODO: implement pax format
- TODO: implement post-processing scripting

Questions / Comments

Thank you for your attention!

https://linutronix.de/minicoredumper

<john.ogness@linutronix.de>