LINUTRONIX
LINUX FOR INDUSTRY

Creating Efficient Small Core Dumps for Embedded Systems

John Ogness

Linutronix GmbH

2016-10-12

LINUTRONIX
LINUX FOR INDUSTRY

What are core dumps?

```
$ man 5 core

CORE(5)                 Linux Programmer's 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 disk 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.

LINUTRONIX
LINUX FOR INDUSTRY

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

LINUTRONIX
LINUX FOR INDUSTRY

The minicoredumper Project

Primary Goals

- minimal core dumps
- custom core dumps
- state snapshots

Main Components

- minicoredumper
- libminicoredumper
- live dumps

LINUTRONIX
LINUX FOR INDUSTRY

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

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 kernel 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 a program to be
       executed. Instead of being written to a disk file, the core
       dump is given as standard input to the program.
```

LINUTRONIX
LINUX FOR INDUSTRY

/proc/sys/kernel/core_pattern

Inform the kernel to use the minicoredumper to handle core dumps, specifying how it is called.

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

$ 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   executable filename (without path prefix)
```

LINUTRONIX
LINUX FOR INDUSTRY

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

LINUTRONIX
LINUX FOR INDUSTRY

minicoredumper.cfg.json

Configuration file example:

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

LINUTRONIX
LINUX FOR INDUSTRY

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",
            "follow_ptr": true,
            "data_len": 42
        }
    ],
    "compression": {
        "compressor": "gzip",
        "extension": "gz",
        "in_tar": true
    },
    "write_proc_info": true
}
```

LINUTRONIX
LINUX FOR INDUSTRY

How It Works

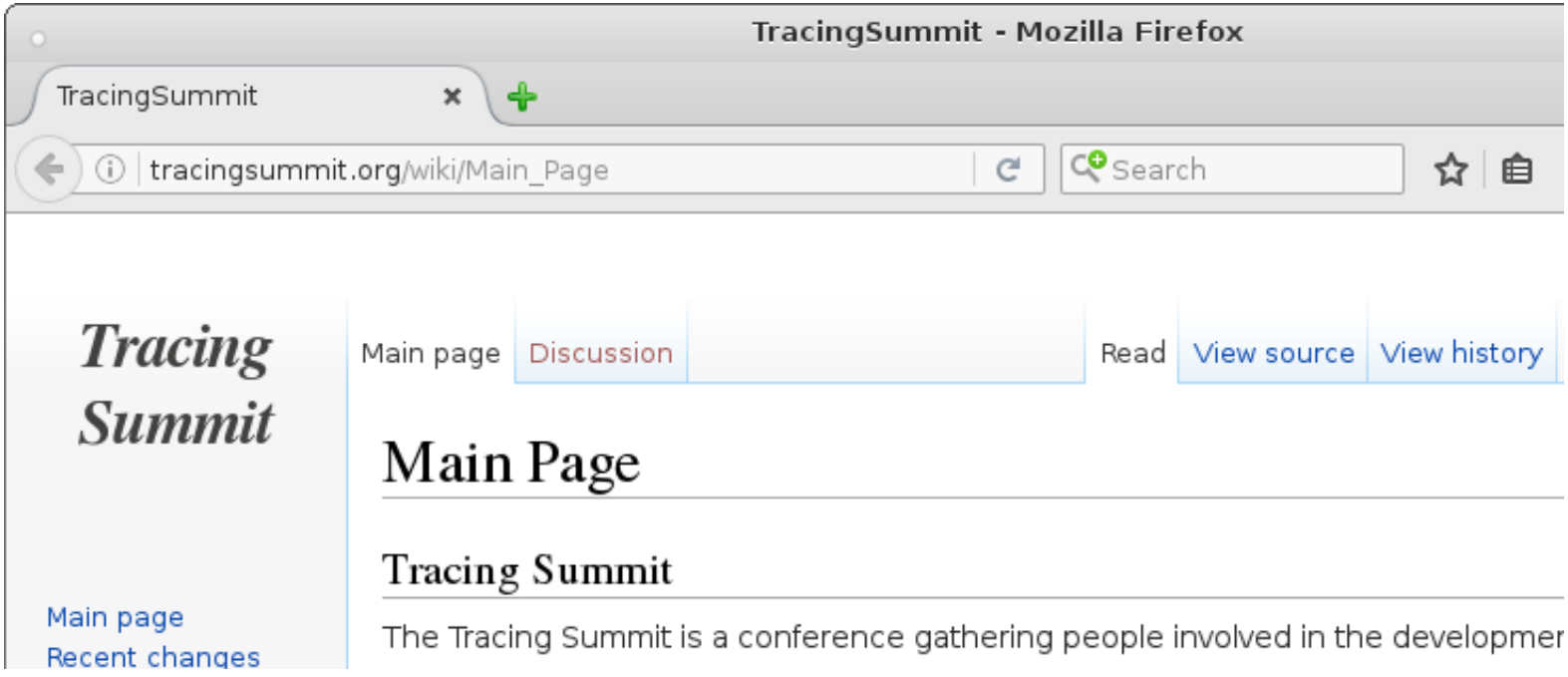
identify process data

- 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 -SEGV `pidof firefox-esr`
```

LINUTRONIX
LINUX FOR INDUSTRY

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.gz
default	523,300 KB	143,228 KB	28,286 KB
minicore/*	526,380 KB	7,928 KB	1,336 KB
minicore/1	522,412 KB	724 KB	31 KB

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

LINUTRONIX
LINUX FOR INDUSTRY

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 2020b14c 00000030
[ 2] .debug                            PROGBITS  00000000 00008540 20201ac0
[ 3] .note.minicoredumper.dumplist     NOTE      00000000 2020a000 0000114c

[...]

Note section [ 3] '.note.minicoredumper.dumplist' of 4428 bytes
at offset 0x2020a000:
  Owner          Data size  Type
  minicoredumper 4400       <unknown>: 80
```

LINUTRONIX
LINUX FOR INDUSTRY

gdb Support

Non-dumped data always has a value of zero because of the sparse core.

```
$ gdb /usr/bin/firefox-esr core
[...]
(gdb) print _edata
$1 = 0
```

A proof-of-concept gdb fork to interpret the custom ELF section note is available:

https://github.com/Linutronix/binutils-gdb/
(branch: minicoredumper-section-note)

```
$ gdb-linutronix /usr/bin/firefox-esr core
[...]
(gdb) print _edata
$1 = <unavailable>
```

LINUTRONIX
LINUX FOR INDUSTRY

Dependencies

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

```
$ objdump -x /usr/sbin/minicoredumper | grep NEEDED

  NEEDED               libelf.so.1
  NEEDED               libjson-c.so.2
  NEEDED               libthread_db.so.1
  NEEDED               libpthread.so.0
  NEEDED               librt.so.1
  NEEDED               libc.so.6
```

LINUTRONIX
LINUX FOR INDUSTRY

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...

LINUTRONIX
LINUX FOR INDUSTRY

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

LINUTRONIX
LINUX FOR INDUSTRY

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/x86_64-linux-gnu/libminicoredumper.so.2.0.0 \
          | grep '\sDO\s'

00201c40 g DO .data 00000004 Base mcd_dump_data_version
00201cc8 g DO .bss  00000008 Base mcd_dump_data_head
```

LINUTRONIX
LINUX FOR INDUSTRY

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);
```

LINUTRONIX
LINUX FOR INDUSTRY

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! */
```

LINUTRONIX
LINUX FOR INDUSTRY

Example Application Debugging

```
$ ./mycrasher
Segmentation fault (core dumped)
$ sudo chown -R `id -u` /.../mycrasher.20161012.093000+0200.19481
$ cd /.../mycrasher.20161012.093000+0200.19481
$ find . -type f
./dumps/19481/i.bin
./dumps/19481/out.txt
./core.tar.gz
./symbol.map
```

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

```
$ cat dumps/19481/out.txt
s="my string" *i=42
```

Example Application Debugging (cont)

```
$ tar -xzSf core.tar.gz

$ gdb-linutronix /.../mycrasher core

[...]

Core was generated by `./mycrasher'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000000004008d2 in main () at mycrasher.c:26
26                  *x = 0;
(gdb) print s
$1 = 0x11eb010 "my string"
(gdb) print i
$2 = (int *) 0x11eb030
(gdb) print *i
$3 = <unavailable>
```

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/19481/i.bin

injected: i.bin, 4 bytes, direct

$ gdb-linutronix /.../mycrasher core

[...]

Core was generated by `./mycrasher'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x00000000004008d2 in main () at mycrasher.c:26
26                  *x = 0;
(gdb) print s
$1 = 0x11eb010 "my string"
(gdb) print i
$2 = (int *) 0x11eb030
(gdb) print *i
$3 = 42
```

LINUTRONIX
LINUX FOR INDUSTRY

Dependencies

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

```
$ objdump -x /usr/lib/x86_64-linux-gnu/libminicoredumper.so.2.0.0 \
          | grep NEEDED

  NEEDED               libc.so.6
```

LINUTRONIX
LINUX FOR INDUSTRY

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...

LINUTRONIX
LINUX FOR INDUSTRY

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

LINUTRONIX
LINUX FOR INDUSTRY

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

```
$ netstat | grep minicoredumper

unix  2  [ ]  DGRAM  61620 @minicoredumper.24111
unix  2  [ ]  DGRAM  61619 @minicoredumper

$ ls -l /dev/shm/minicoredumper.shm

-rw------- 1 mcd mcd 56 Oct 12 09:30 /dev/shm/minicoredumper.shm
```

LINUTRONIX
LINUX FOR INDUSTRY

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

LINUTRONIX
LINUX FOR INDUSTRY

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

LINUTRONIX
LINUX FOR INDUSTRY

Dependencies

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

```
$ objdump -x /usr/sbin/minicoredumper_regd | grep NEEDED

  NEEDED               libpthread.so.0
  NEEDED               librt.so.1
  NEEDED               libc.so.6
```

LINUTRONIX
LINUX FOR INDUSTRY

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

LINUTRONIX
LINUX FOR INDUSTRY

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

LINUTRONIX
LINUX FOR INDUSTRY

Project Status

- about to release version 2.0.0 (presented here)
- working on packaging for Debian/Stretch
- working on Yocto layer for OpenEmbedded

LINUTRONIX
LINUX FOR INDUSTRY

Questions / Comments

Thank you for your attention!

`https://linutronix.de/minicoredumper`

`RCPT TO:<john.ogness@linutronix.de>`

LINUTRONIX
LINUX FOR INDUSTRY

Linutronix GmbH

Bahnhofstr. 3
88690 Uhldingen-Muehlhofen