Debugging for production systems

February, 2013

Tristan Lelong
Adeneo Embedded
tlelong@adeneo-embedded.com

Embedded Linux Conference 2013
Who am I?

- Software engineer at Adeneo Embedded (Bellevue, WA)
- Systems:
  - GNU/Linux
  - Android
- Main activities:
  - BSP adaptation
  - Driver development
  - System integration
Production systems constraints

- Production systems have limited resources
- Production systems are secured
- Production systems are not connected
- Production systems are not accessible
Production systems constraints

- Production systems have limited resources
  - Limited CPU
    - Cannot do some heavy processing
  - Limited RAM
    - Cannot process huge files
  - Limited ROM
    - Cannot store all symbols, debug tools
Production systems constraints

- Production systems are secured
  - No external access to the filesystem
  - No automatic information reporting
  - No tools to perform in-depth analysis
Production systems constraints

- Production systems are not connected
  - No internet connection to send reports
  - No link to do remote debugging
Production systems constraints

- Production systems are not accessible
  - No UI for end users
  - Not in the developers' hands
Production systems constraints

• Some more constraints
  • A production system is not only “your software”
    – No full knowledge of all components
    – No control on other components
    – Build systems tend to give more control
• ...
Purpose of this presentation

• Tim Bird did a very good presentation about debugging in production systems during ELC 2012 (Appropriate Crash Handling in Linux)

• This presentation focuses on
  • The same subject
  • Other key points
  • Further information
Purpose of this presentation

- This subject concerns everybody
- This is a key point in developing a product
- Not much information about it
- Sometimes postponed until last minute...
About this presentation

- Focus on stock systems
- Use of existing components
- Adding new components
Tools: Linux kernel coredumps

- Linux kernel offers a coredump feature
  - What is a coredump
  - How are they generated
  - What is the typical usage
  - How are coredumps designed
  - Why coredumps may not be suitable
Tools: Linux kernel coredumps

• What is a coredump
  • Process has a virtual memory space
  • Memory is divided in segment
    – Code: software / libraries
    – Stack
    – Heap
  • Segments are mapped in the process virtual address space
Tools: Linux kernel coredumps

- What is a coredump
  - Software runs in userspace
    - Linux kernel enforces permissions
  - When an error condition is detected, the kernel notify the software using unix signals.
    - SIGSEGV
    - SIGBUS
    - SIGABRT
    - SIGFPE
    - SIGTRAP
    - SIGILL
    - SIGBUS
    - SIGQUIT
    - SIGXCPU
    - SIGSYS
    - SIGXFSZ
Tools: Linux kernel coredumps

- Man 7 signal
  - All signals
  - Signum
  - Short description
  - Default actions
  - Much more...
Tools: Linux kernel coredumps

- How are they generated: user side
  - Need to activate the ELF_CORE option in kernel
  - Need to set the core_pattern
    - `$> echo “core” > /proc/sys/kernel/core_pattern`
  - Need to set ulimit
    - `$> ulimit -c unlimited`
- Special care for busybox systems
  - Activate: FEATURE_INIT_COREDUMPS
  - A special file .init_enable_core must be present in /
Tools: Linux kernel coredumps

• How are they generated: kernel side
  • When delivering a signal: `get_signal_to_deliver()`
  • If signal matches the mask: `SIG_KERNEL_COREDUMP_MASK`
  • Kernel calls `do_coredump()` from the filesystem subsytem
    – Various checks (recursive crash, command format, ...)
    – Open the output descriptor (file or pipe)
  • Then calls `core_dump()` for the current binary format
    – `elf_core_dump()` is called for ELF
    – Collect all data and dump segments to the output descriptor (file or pipe)
Tools: Linux kernel coredumps

- What is the typical usage
  - Debugging with GDB
  - Using the symbols from debug binaries
  - Full access to backtrace, variables
Tools: Linux kernel coredumps

DEMO

GDB + coredump for post-mortem analysis
Tools: Linux kernel core dumps

• How are coredumps designed:
  • Coredumps are ELF files
  • The ELF e_type is ET_CORE (4)
ELF file format

- **Executable and Linkable Format**
- Reference in UNIX systems since 1999
- Standardized structure:
  - Headers
  - Segments (physical view)
  - Sections (linker view)
ELF file format

- On GNU/Linux memory map are reachable:
  - /proc/<pid>/maps

```bash
$ cat /proc/32026/maps
08048000-08129000 r-xp 00000000 08:01 10888084 /bin/bash
08129000-0812a000 r-p 00000000 08:01 10888084 /bin/bash
0812a000-0812f000 rw-p 00000000 08:01 10888084 /bin/bash
0812f000-08134000 rw-p 00000000 00:00 0
09627000-098e0000 rw-p 00000000 00:00 0 [heap]
b7439000-b745b000 r-p 00000000 08:01 183354 /usr/share/locale/fr/LC_MESSAGES/bash.mo
b745b000-b7465000 r-xp 00000000 08:01 2265084 /lib/i386-linux-gnu/i686/cmov/libnss_flies-2.13.so
...
b7490000-b7491000 rw-p 00006000 08:01 2265072 /lib/i386-linux-gnu/i686/cmov/libnss_compat-2.13.so
b7491000-b7609000 r-p 00000000 08:01 196268 /usr/lib/locale/locale-archive
b7609000-b760a000 rw-p 00000000 00:00 0
b760a000-b7760000 r-xp 00000000 08:01 2265083 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
...
b7763000-b7764000 rw-p 00158000 08:01 2265083 /lib/i386-linux-gnu/i686/cmov/libc-2.13.so
b7764000-b7768000 rw-p 00000000 00:00 0
b7768000-b776a000 r-xp 00000000 08:01 2265066 /lib/i386-linux-gnu/i686/cmov/libdl-2.13.so
...
b778b000-b778c000 rw-p 0001e000 08:01 542 /lib/i386-linux-gnu/libc.so.5.9
b77a8000-b77af000 r-s 00000000 08:01 646575 /usr/lib/i386-linux-gnu/gconv/gconv-modules.cache
...
b77ce000-b77cf000 r-p 0001b000 08:01 655 /lib/i386-linux-gnu/gconv-modules.cache
b77cf000-b77d0000 rw-p 0001c000 08:01 655 /lib/i386-linux-gnu/gconv-modules.cache
bf868000-bf889000 rw-p 00000000 00:00 0 [stack]
```
ELF file format

• Man elf
  • Headers description
  • Segments and sections format
  • Description of all structures members
  • Listing / description of standard sections
Tools: Linux kernel coredumps

- How are coredumps designed
  - Generic ELF header
  - Description of all segments
  - No section / No symbol
  - One specific segment: PT_NOTE
  - Text segment
  - Stack segment (end of memory space)
Tools: Linux kernel coredumps

• Why coredumps may not be suitable
  • Coredumps are binaries: need tools to interpret
  • Coredumps are large
    – All segment dumped: main app + libraries
    – Multithread increases the overall size (8MB / thread)
  • Coredump does not contain debug symbols
Tools: binutils / objdump

- Specific to the architecture
- Part of the toolchain
- Contains useful tool for debugging
  - objdump
Tools: binutils / objdump

- Objdump parses an ELF file
  - Reads headers
  - Reads segments / sections tables
  - Dumps segments / sections
  - Disassembles code
  - Resolves symbols and debugging information
Tools: binutils / objdump

DEMO
structure of a coredump using objdump
Tools: Linux kernel coredumps

- The PT_NOTE segment is not a memory dump
  - It is generated by Linux kernel function `fill_note_info()`
  - It is then dumped as the first segment of coredump
  - It contains generic process info
Tools: Linux kernel coredumps

- **PT_NOTE segment:**
  - prstatus: NT_PRSTATUS (for each thread)
    - Signals / pids / time / registers
  - psinfo: NT_PRPSINFO
    - Process state / UID / GID / name / args
  - auxv: NT_AUXV
  - fpu: NT_PRFPREG
  - xfpu: ELF_CORE_XFPREG_TYPE
Tools: libc

- Libc offers an unwinding function:
  - `backtrace()`
- It can even resolve addresses to symbols
  - `backtrace_symbols()`, `backtrace_symbols_fd()`
- Work only inside the current process
  - Useful for error logs in your software
Tools: ptrace

- ptrace is a system call in unix systems
  - IRIX, AIX, NetBSD, FreeBSD, OpenBSD, Linux...
- Access to another process memory space
  - Registers, data, code
- Very powerful
  - For debugging
  - For profiling
  - For on the fly patching
Tools: libunwind

- Libunwind project
  - Aims at providing a common API to determine the call chain of a program.
  - Works locally and remotely (need to use accessors).
  - Works with several architectures:
    - x86, x86_64, ppc (32/64), mips, ia64, hppa, arm, superH
- Since release 1.1 (oct 2012): can unwind coredumps
  - Still need the full coredump file
Tools: gdb

- GNU debugger
- Really powerful
  - Remote debugging
  - Scripting for auto debugging
Tools: debuggerd

- Android debugging service
  - Debuggerd is running in background
  - Uses a custom libc: android bionic to hook signal reception
  - Uses ptrace to access process information
  - Good unwinding but working only for ARM
Tools: crash_handler

- Tim Bird crash handler
- Works for ARM architecture
- Hooks on core generation: core_pattern
- Uses ptrace and /proc for information extraction
- Unwinding:
  - Best guess
  - Based on unwinding tables
  - McTernan: modeling an ARM processor
Tools: cortex

- Author: Tristan Lelong
- Started in 2011
- Objective was to have a userland equivalent to kernel oopses
  - Generate crash report with relevant informations in a simple and human (developer) readable format.
Tools: cortex

- The name comes from the contraction of coredump and textreport: **COReTEXt**
- Cortex aims at converting binary cores to text reports.
Tools: cortex

- Dependency on binutils for code disassembly
- No dependency on libelf
  - Use only standard elf.h header libc
  - Easy raw access to the segments
  - Parsing of core is done on-the-fly to comply with the core_pattern streaming feature: not seekable stream.
Tools: cortex

- Easily integrated in the target system
  - Installed as a core handler in core_pattern proc file
  - Can be integrated in a core handling chain
  - No system source code modification required
Tools: cortex

- Architectures are handled in separate modules
- Makefile selects the architecture to include depending on configure variables
- New architectures can be added by following a standardized API
  - declared in arch/cortex_arch.h
  - 7 functions exported in struct cortex_arch_ops
Tools: cortex

DEMO
cortex text report generation
Tools: cortex

• Constraints when using cortex
  • The unwinding is relying on frame pointers (option -fno-omit-framepointer required for accurate results)
  • Equivalent production binaries with symbols must be generated and kept.
Tools: cortex

• Another feature recently added is the coredump stripping:
  • Remove all text sections except the current one
  • Keep only the top part of the stack segment

• Stripped coredumps are compatible with gdb even though less information is available
Tools: cortex

DEMO
cortex stripped coredump
Tools: cortex

- Google project already online
  - http://cortex-tool.googlecode.com
  - Sources on git
  - GPLv2 licensed
- First release: cortex-0.1 is available
  - Handle ARM, x86 architectures
  - Basic unwinding based on frame pointers
Tools: cortex

- Next steps:
  - Improve unwinding
  - Add new architectures
  - Include it in build systems
Debugging for production systems

Thank you for your attention

Questions?