Fosdem 2015

perf status on ARM and ARM64

jean.pihet@newoldbits.com
Contents

• Introduction
  • Scope of the presentation
  • Supported tools
• Call stack unwinding
  • General
  • Methods
  • Corner cases
• ARM and ARM64 support
• Next steps, follow-up
• References
Introduction

- Scope of the presentation
  - Work done for Linaro LEG:
    - profiling tools for servers load,
    - features parity with x86.
  - This presentation is about the call stack unwinding on ARM/ARM64, using fp and dwarf methods.
- Tool in use: perf
Call stack unwinding

- General
  - perf tool regularly captures (perf record) the current state and then parses the data (perf report).
  - perf links with unwinding libraries.
  - Unwinding allows to trace the callers up to the current execution point.

- Example: The 'stress_bt' application consists of a long call chain (foo_1 calling foo_2 calling ... foo_128). foo_128 performs some calculation on u64 variables. The main loop calls foo_1, foo_2 ... foo_128 in order.

- Without and with unwinding:
Call stack unwinding

• Without and with unwinding:

```
# perf record

usage: perf record [options] [command]
or: perf record [options] -- <command> [options]
...
-g                enables call-graph recording
--call-graph <mode[,dump_size]>
    setup and enables call-graph (stack chain/backtrace)
recording: fp dwarf

# perf record -- ./stress_bt
# perf report
98.34% stress_bt stress_bt 
0.11% stress_bt stress_bt 
0.10% stress_bt libc-2.17-2013.07-2.so 
0.08% stress_bt stress_bt 
0.07% stress_bt stress_bt 
...
0.01% stress_bt [kernel.kallsyms] 
0.01% stress_bt [kernel.kallsyms] 
0.01% stress_bt stress_bt 
0.01% stress_bt stress_bt 
0.01% stress_bt stress_bt 
0.01% stress_bt stress_bt 
0.01% stress_bt stress_bt 
...
```

```
# perf record --call-graph dwarf -- ./stress_bt
# perf report (--call-graph --stdio)
96.93% stress_bt stress_bt 
  |   foo_128
  --- foo_128
  |---98.22% foo_127
  |     |---99.46% foo_126
  |     |     |---99.11% foo_125
     |     |---0.89% bar
     |     |     doit
     |     |     main
     |     |     __libc_start_main
     |     ...
  |---0.77% bar
     | doit
     | main
     |     __libc_start_main
     |---1.01% [...]
0.25% stress_bt [kernel.kallsyms] 
  | page_mkclean
  --- page_mkclean
```
Call stack unwinding

• General
  • There are different methods to allow the use of call stack unwinding.
  • Support is needed from:
    − Compiler + compilation options,
    − kernel arch code,
    − perf tool + external libraries (libunwind, libdw).

• Methods
  • .exidx
  • frame pointer
  • dwarf
Call stack unwinding

- Method: `.exidx`
  - Unwinding info stored in specific ELF sections `.ARM.exidx` and `.ARM.extab`.
  - Generated by GCC under `-funwind-tables` and `-fasynchronous-unwind-tables`.
  - No change -so no overhead- to the code.
  - Overhead to the binary size.
  - Supported by libunwind on ARM.
  - Not supported by perf.
Call stack unwinding

- Method: frame pointer
  - Defined by the ABI
  - During execution the context is stored on the stack as a linked list of stack frames. fp is the frame pointer.
    \[ fp = \text{old } sp, \text{ similar to } lr = \text{old } pc. \]
  - Generated by GCC under `-fno-omit-frame-pointer`. Not enabled by default.
  - Code overhead for the stack handling, code size overhead.
Call stack unwinding

- **Method: frame pointer**

  ; Prologue - setup

  mov   ip, sp                    ; get a copy of sp.
  stm   sp!, {fp, ip, lr, pc}    ; Save the frame on the stack.
  sub   fp, ip, #4               ; Set the new frame pointer.

  ...

  ; Function code comes here

  ; Could call other functions from here

  ...

  ; Epilogue - return

  ldm   sp, {fp, sp, lr}         ; restore stack, frame pointer and old link.
  bx    lr                        ; return.
Call stack unwinding

- Method: dwarf
  - Unwinding info stored in specific ELF section `.debug_frame`.
  - Platform independent format.
  - Generated by GCC under `-g`.
  - Overhead only to the debug binary size.
  - On most distros the `-dbg` flavor of the libraries in `/usr/lib/debug/lib` usually contain the correct debug information.
  - No change -so no overhead- to the code.
Call stack unwinding

• Method: dwarf

```c
# dwarfdump -f -kf stress_bt .debug_frame

fde:
<  0><0x0000842c:0x00008498><foo_128><fde offset 0x00000010 length:
0x00000014><eh offset none>
  0x0000842c: <off cfa=00(r13) >
  0x0000842e: <off cfa=04(r13) > <off r14=-4(cfa) >
  0x00008430: <off cfa=24(r13) > <off r14=-4(cfa) >
<  0><0x00008498:0x000084a4><foo_127><fde offset 0x00000028 length:
0x00000014><eh offset none>
  0x00008498: <off cfa=00(r13) >
  0x0000849a: <off cfa=08(r13) > <off r3=-8(cfa) > <off r14=-4(cfa) >
...
<  0><0x00008ccc:0x00008cf2><main><fde offset 0x00000c40 length:
0x00000014><eh offset none>
  0x000008cc: <off cfa=00(r13) >
  0x000008cce: <off cfa=04(r13) > <off r14=-4(cfa) >
  0x000008cd0: <off cfa=16(r13) > <off r14=-4(cfa) >
```

cie:
<  0> version 1
  cie section offset 0 0x00000000 augmentation
  code_alignment_factor 2
  data_alignment_factor -4
  return_address_register 14
  bytes of initial instructions 3
cie length 12
initial instructions
  0 DW_CFA_def_cfa r13 0
Call stack unwinding

• Gotchas (= Corner Cases)
  • 32-bit compatibility mode
    – A 32-bit ARM binary can run on ARM64.
    – The unwinding on ARM64 has to correctly handle the 32-bit structs (registers, fp struct, dwarf info...).
    – The impact is on all components (kernel, perf, libraries etc.).
Call stack unwinding

• Gotchas (= Corner Cases)
  • tail call optimization
    – No code for the stack frame handling for a tail call.
    – Confuses the fp based unwinding.
    – Dwarf info encodes the call chain.
    – Need more check/test.

```c
void bar(int val)
{
    printf("Meet @ bar\n");
    return;
}

void foo(int val)
{
    bar(x);
    return;
}

int main()
{
    foo(42);
    return 0;
}
```
Call stack unwinding

- **Gotchas (=Corner Cases)**
- ARM assembly directives
  - Example: generic register used as link register.
  - It seems that dwarf correctly encodes the info but unwinding is not OK.
  - Need more check/test

```c
ENTRY(__kernel_gettimeofday)
  .cfi_startproc
  mov    x2, x30
  .cfi_register x30, x2
  /* Acquire the sequence counter and get the timespec */
  adr    vdso_data, _vdso_data
  1:     seqcnt_acquire
  cbnz   use_syscall, 4f
  ...
  ret    x2
  .cfi_endproc
ENDPROC(__kernel_gettimeofday)
```
ARM and ARM64 support

- Kernel arch code
- perf code + test suite
- External libraries

<table>
<thead>
<tr>
<th></th>
<th>arch: fp</th>
<th>arch: dwarf</th>
<th>perf: libunwind</th>
<th>perf: libdw</th>
<th>Perf: test suite</th>
<th>Compat mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ARM64</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x submitted</td>
<td>x submitted</td>
<td>✓</td>
</tr>
</tbody>
</table>
Next steps, follow-up

- Submitted patches, to check
  - Generic: tracing with kernel tracepoints events
    https://lkml.org/lkml/2014/7/7/282
  - ARM64 libdw
    https://lkml.org/lkml/2014/5/6/395
  - ARM64 test suite
    https://lkml.org/lkml/2014/5/6/392
    https://lkml.org/lkml/2014/5/6/398
- Tail call optimization: to check
- ARM directives: to check
- .exidx support in perf?
References

- Details on libunwind and .exidx unwinding: https://wiki.linaro.org/KenWerner/Sandbox/libunwind
- Dwarf unwinding details: https://wiki.linaro.org/LEG/Engineering/TOOLS/perf-callstack-unwinding
- libunwind: http://www.nongnu.org/libunwind/
- libdw/elfutils: https://fedorahosted.org/elfutils/
- ARM directives: http://sourceware.org/binutils/docs/as/ARM-Directives.html
- LKML and linux-arm-kernel MLs
- perf IRC channel: #perf at irc.oftc.net
Questions?

Thank you!
Back-up slides

• perf compilation. ! Use a recent libunwind dev library!

```
$ make LIBUNWIND_DIR=/usr/local NO_LIBDW_DWARF_UNWIND=1 -C tools/perf

... Auto-detecting system features:
...    dwarf: [ on ]
...    glibc: [ on ]
...    gtk2: [ OFF ]
...    libaudit: [ on ]
...    libbfd: [ on ]
...    libelf: [ on ]
...    libnuma: [ OFF ]
...    libperl: [ on ]
...    libpython: [ on ]
...    libslang: [ on ]
...    libunwind: [ on ]
...    libdw-dwarf-unwind: [ on ]
...    zlib: [ on ]

...    DWARF post unwind library: libunwind

$ make LIBDW_DIR=/usr/local NO_LIBUNWIND=1 -C tools/perf
```