Stack frame unwinding on ARM

Ken Werner

LDS, Budapest 2011

http:/www.linaro.org

Linaro

why?

- Who needs to unwind the stack?
  - C++ exceptions
  - GDB
  - anyone who wants to display the call chain

Unwinding in General

- How is this possible?
  - ABI's do not specify a dedicated frame pointer
  - but there are special sections:

```
$ echo "void foo(){}" > foo.c
$ gcc -funwind-tables -c foo.c
$ readelf -S foo.o
There are 11 section headers, starting at offset 0xf4:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000005 00  AX  0   0  4
  [ 2] .data             PROGBITS        00000000 00003c 000000 00  WA  0   0  4
  [ 3] .bss              NOBITS          00000000 00003c 000000 00  WA  0   0  4
  [ 4] .comment          PROGBITS        00000000 00003c 00002b 01  MS  0   0  1
  [ 5] .note.GNU-stack   PROGBITS        00000000 000067 000000 00      0   0  1
  [ 6] .eh_frame         PROGBITS        00000000 000068 000038 00   A  0   0  4
  [ 7] .rel.eh_frame     REL             00000000 000348 000008 08      9   6  4
  [ 8] .shstrtab         STRTAB          00000000 0000a0 000053 00      0   0  1
  [ 9] .symtab           SYMTAB          00000000 0002ac 000090 10     10   8  4
  [10] .strtab           STRTAB          00000000 00033c 00000b 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Unwinding on ARM

- ... is different
  - no .eh_frame but .ARM.exidx and .ARM.extab:

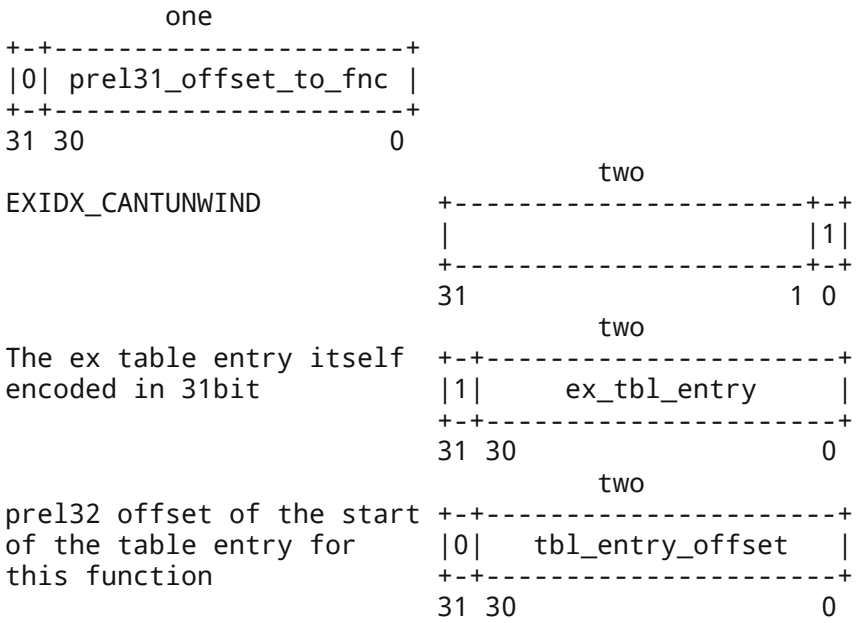
```
$ echo "void foo(){}" > foo.c
$ arm-linux-gnueabi-gcc -funwind-tables -c foo.c
$ readelf -S foo.o
There are 13 section headers, starting at offset 0x118:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 00000c 00  AX  0   0  4
  [ 2] .data             PROGBITS        00000000 000040 000000 00  WA  0   0  1
  [ 3] .bss              NOBITS          00000000 000040 000000 00  WA  0   0  1
  [ 4] .ARM.extab        PROGBITS        00000000 000040 000000 00   A  0   0  1
  [ 5] .ARM.exidx        ARM_EXIDX       00000000 000040 000008 00  AL  1   0  4
  [ 6] .rel.ARM.exidx    REL             00000000 000428 000010 08     11   5  4
  [ 7] .comment          PROGBITS        00000000 000048 00002b 01  MS  0   0  1
  [ 8] .note.GNU-stack   PROGBITS        00000000 000073 000000 00      0   0  1
  [ 9] .ARM.attributes   ARM_ATTRIBUTES  00000000 000073 000033 00      0   0  1
  [10] .shstrtab         STRTAB          00000000 0000a6 00006f 00      0   0  1
  [11] .symtab           SYMTAB          00000000 000320 0000e0 10     12  12  4
  [12] .strtab           STRTAB          00000000 000400 000028 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

.ARM.exidx

- .ARM.exidx contains a sorted list of key-value pairs
- 'key' is a prel31 offset to the start of a function
- 'value' contains one of the following three formats:
  - bit 31 is zero: this is a prel31 offset of the start of the table entry for this function
  - bit 31 is one: this is a table entry itself
  - set to 0x1: the function cannot be unwound

.ARM.exidx

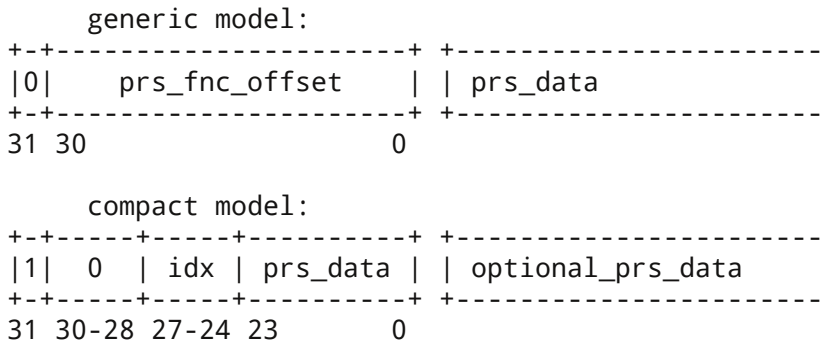
- the index table entry consists of two words:



.ARM.extab

- generic model
- compact model

(bit 31 of the first word is set - in this case the bits 24-27 selects one of the default personality routines.)



personality routines on ARM

- there are three routines defined on ARM:

index	name	description
0	Su16 / __aeabi_unwind_cpp_pr0	Short frame unwinding description followed by descriptors with 16-bit scope
1	Lu16 / __aeabi_unwind_cpp_pr1	Long frame unwinding description followed by descriptors with 16-bit scope
2	Lu32 / __aeabi_unwind_cpp_pr1	Long frame unwinding description followed by descriptors with 32-bit scope

The short frame unwinding routine (Su16) expects a fixes set of three unwind instructions encoded into bits 0-23. The long frame unwinding routine (Lu16) encode the number of unwind instructions in bits 16-23. The .ARM.extab section can be omitted if the "compact model" is used in conjunction with the short frame unwinding description:

```
short:
+-+-----+-----+----------+
|1|  0  |  0  | prs_data |
+-+-----+-----+----------+
31 30-28 2724/ 23       0 \
            /             \
            +----+----+----+
            |insn|insn|insn|
            +----+----+----+
            23-16 15-8 7 - 0
```

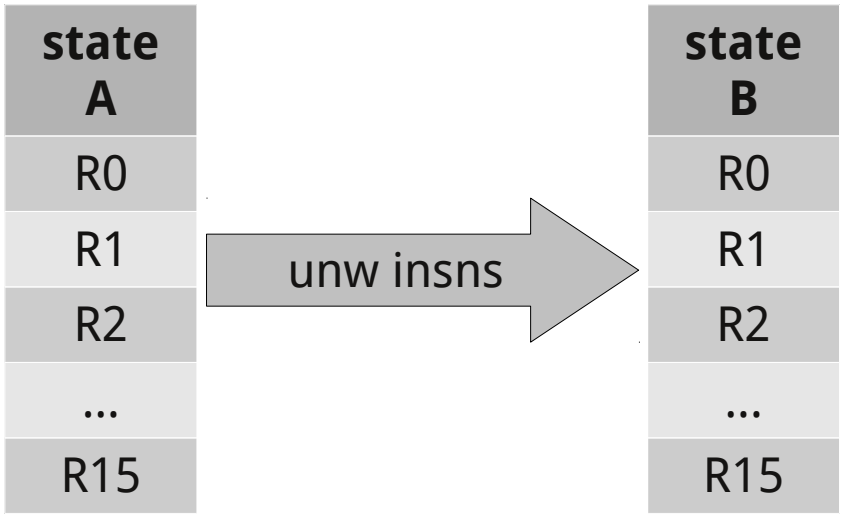
```
long:
+-+-----+-----+----------+ +----------------------
|1|  0  | 1or2| prs_data | |prs_data
+-+-----+-----+----------+ +----------------------
31 30-28 2724/ 23       0                          \
            /                                       \
            +----+----+----++----+----+----+----+---
            |  N |insn|insn||insn|insn|insn|insn|...
            +----+----+----++----+----+----+----+---
            23-16 15-8 7 - 0
With 'N' specifying the number of additional insn
```

Linaro

frame unwinding instructions

Instruction	Mnemonic	description
00xxxxxx	ARM_EXIDX_CMD_DATA_POP	vsp = vsp + (xxxxxx << 2) + 4. Covers range 0x04-0x100 inclusive
01xxxxxx	ARM_EXIDX_CMD_DATA_PUSH	vsp = vsp – (xxxxxx << 2) - 4. Covers range 0x04-0x100 inclusive
10000000 00000000	ARM_EXIDX_CMD_REFUSED	refuse to unwind (0x80 followed by 0x00)
1000iiii iiiiiiii	ARM_EXIDX_CMD_REG_POP	Pop up to 12 integer registers under masks {r15-r12}, {r11-r4}
1001nnnn	ARM_EXIDX_CMD_REG_TO_SP	Set vsp = r[nnnn]
10100nnn	ARM_EXIDX_CMD_REG_POP	Pop r4-r[4+nnn]
10101nnn	ARM_EXIDX_CMD_REG_POP	Pop r4-r[4+nnn], r14
10110000	ARM_EXIDX_CMD_FINISH	Finish
10110001 0000iiii	ARM_EXIDX_CMD_REG_POP	Pop integer registers under mask {r3, r2, r1, r0}
10110010 uleb128	ARM_EXIDX_CMD_DATA_POP	vsp = vsp + 0x204 + (uleb128 << 2)
10110011 ssssqqqq	ARM_EXIDX_CMD_VFP_POP	Pop VFP double-precision registers D[ssss]-D[ssss+qqqq]
10111nnn	ARM_EXIDX_CMD_VFP_POP	Pop VFP double-precision registers D[8]-D[8+nnn]
11010nnn	ARM_EXIDX_CMD_VFP_POP	Pop VFP double-precision registers D[8]-D[8+nnn]
11000nnn	ARM_EXIDX_CMD_WREG_POP	Intel Wireless MMX pop
11000111 0000iiii	ARM_EXIDX_CMD_WCGR_POP	Intel Wireless MMX pop WCGR registers under mask {wCGR3,2,1,0}
all other	ARM_EXIDX_CMD_RESERVED	Reserved or Spare

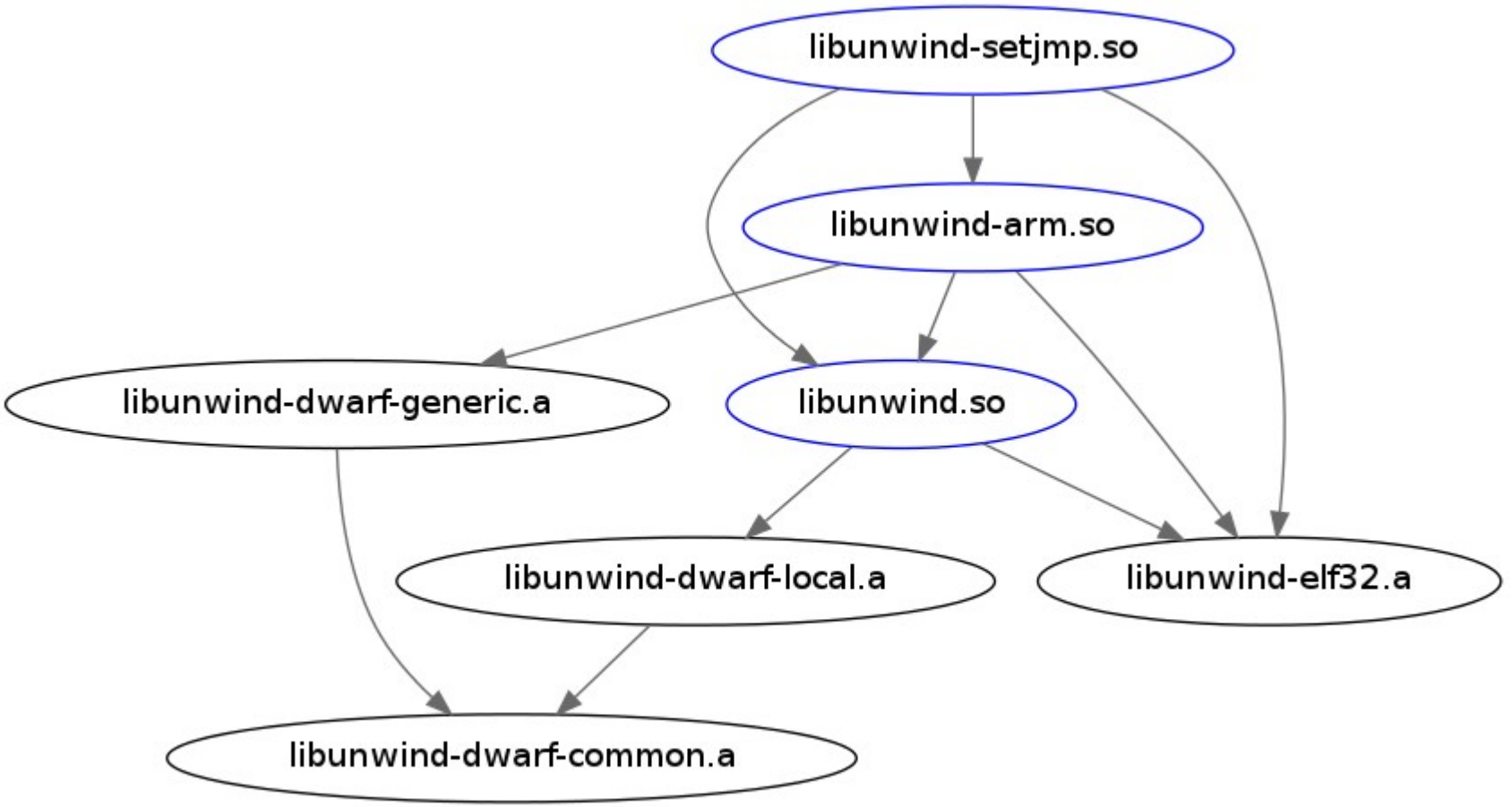
machine state



libunwind

- allows you to easily walk the stack frames
- access to the callee-saved registers contents
- support for resuming execution at a certain frame

libunwind



libunwind library dependencies

Linaro

libunwind

- Demo!

libunwind

- What did we achieve in the last few months?
  - support for lookup, extracting and decoding entries from the ARM specific unwind sections
  - support for unwinding call stacks with mixed DWARF- and extbl-frames
  - ARM signal frame detection and handling
  - support for resuming execution at a certain stack frame
  - lots of (build, code, testsuite) fixes
  - and all of it upstream by now

libunwind

- TODO's
  - Continue to investigate test suite failures on ARM (and fix them)
  - handle FP regs
  - implement support for remote unwinding
  - reduce the use of target specific header files in order to simplify cross compiling
  - enhance DWARF support (currently DWARF2 only - if at all)
  - handle the Thumb marker

references

- http://www.nongnu.org/libunwind
- http://git.linaro.org/gitweb?p=people/kwerner/libunwind.git;a=summary
- https://wiki.linaro.org/KenWerner/Sandbox/libunwind
- http://infocenter.arm.com/help/topic/com.arm.doc.ihi0038a/IHI0038A_ehabi.pdf

Linaro