PB173 – Binární programování Linux

VI. DWARF

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

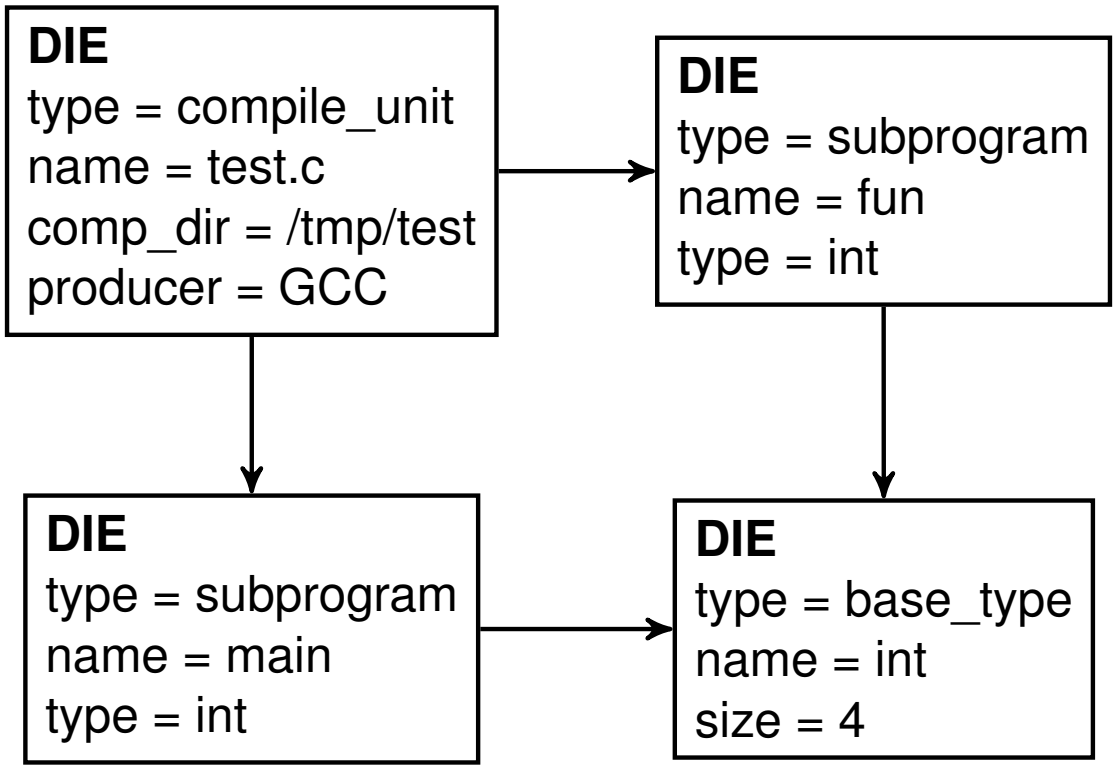
5. 11. 2015

DWARF

- Strukturované ladicí informace
  - Zápis komplikovaných výrazů
  - Vytvářené překladačem
- Nezávislost na jazyku a formátu souborů
- Dokumentace
  - Introduction to the DWARF Debugging Format
  - DWARF Debugging Information Format (Version 4)

DWARF formát

- Ladicí informace v lese skoro-stromů
  - Pro každý zdrojový soubor jeden skoro-strom
- Uzly stromů jsou *Debugging Information Entry* (DIE)
- Jeden DIE může odkazovat na kterýkoliv jiný



DWARF a ELF

1. DWARF je v ELFu uložený v `.debug_*` sekcích
   - `.debug_info`: všechny DIE z lesa stromů
   - `.debug_abbrev`: formát `.debug_info` (Figure 48 v DWARF4)
   - `.debug_line`: překlad instrukce $\leftrightarrow$ řádek kódu
   - `.debug_loc`: popis maker
   - `.debug_str`: řetězce odkazované z `.debug_info`
2. Výpis DWARFu v ELFu
   - `readelf --debug-dump`
   - `objdump --dwarf`
   - `dwarfdump` (z `libdwarf`)

Knihovny na práci s DWARF

Dvě různé implementace knihoven

1. `libdwarf`
   - Starší
   - Hůř se s ní pracuje
   - Obsahuje méně chyb
   - Lépe dokumentovaná
   - `libdwarf/libdwarf.h`
2. `libdw` a `libdwfl`
   - Podpora nových standardů DWARF
   - Lepší rozhraní
   - Základní: `libdw` (`elfutils/libdw.h` + `dwarf.h`)
   - Nadstavba: `libdwfl` (`elfutils/libdwfl.h`)
   - *Tuto budeme používat*

libdw a libdwfl

- Knihovny jsou součástí elfutils
- Lze je používat střídavě (jako `elf` a `gelf`)
- Inicializace: `Dwfl *dwfl_begin(const Dwfl_Callbacks *cb)`
  - `.section_address = dwfl_offline_section_address`
  - `.find_debuginfo = dwfl_standard_find_debuginfo`
- Ukončení: `void dwfl_end(Dwfl *dwfl)`
- Poslední chyba: `dwfl_errmsg(-1)`
- Dokumentace
  - V hlavičkových souborech
- Knihovny při překladu: `gcc ... -ldw`

Úkol

Inicializace `libdwfl`

1. Nainstalujte si `libdw` (a `libdwfl`)
2. Vytvořte si `main`
3. Definujte si statické háčky
   - Struktura `const Dwfl_Callbacks`
   - `.section_address = dwfl_offline_section_address`
   - `.find_debuginfo = dwfl_standard_find_debuginfo`
4. Zavolejte `dwfl_begin`
5. Zavolejte `dwfl_end`
6. Ověřujte návratové hodnoty a vypisujte chyby
7. Přeložte a spusťte

libdw a libdwfl – soubory

Načtení ELFu s DWARFem uvnitř

- `Dwfl_Module *dwfl_report_offline(...)`
  - `Dwfl *dwfl`: návratová hodnota z `dwfl_begin`
  - `const char *name`: pojmenování modulu (např. `file_name`)
  - `const char *file_name`: jméno souboru
  - `int fd`: `-1` (nebo souborový deskriptor, pak `file_name == ""`)

Průchod stromů (jednotlivých Compilation Unit)

- `Dwarf_Die *dwfl_nextcu(...)`
  - `Dwfl *dwfl`: návratová hodnota z `dwfl_begin`
  - `Dwarf_Die *lastcu`: `NULL` nebo předchozí CU
  - `Dwarf_Addr *bias`: přičítá se k adresám, které vrací `libdw`

Příklad

```
Dwfl_Module *mod = dwfl_report_offline(dwfl, argv[1], argv[1], -1);
while ((die = dwfl_nextcu(dwfl, die, &bias))) { ... }
```

Úkol

Otevření ELFu pomocí `libdwfl`

1. Přidávejte kód mezi `dwfl_begin` a `dwfl_end`
2. Zavolejte `dwfl_report_offline`
3. Iterujte přes CU DIE pomocí `dwfl_nextcu`
   - Spočítejte z kolika souborů je objekt přeložen
4. Ověřujte návratové hodnoty a vypisujte chyby
5. Přeložte a spusťte

DIE

Debugging Information Entry

- Typ celého DIE: `DW_TAG_*`
- Z DIE: `int dwarf_tag(Dwarf_Die *die)`
  - `DW_TAG_compile_unit`: DIE o souboru
  - `DW_TAG_subprogram`: DIE o funkci
  - `DW_TAG_base_type`: DIE o typu
  - ...
- Seznam atributů = trojic
  - O co jde: `DW_AT_*`
    - `DW_AT_comp_dir`: adresář, kde se překládalo
    - `DW_AT_name`: jméno (proměnné, souboru, ...)
    - `DW_AT_decl_file`: soubor s deklarací
    - `DW_AT_type`: typ proměnné
    - ...
  - Typ/forma hodnoty (`DW_FORM_*`)
    - `DW_FORM_addr`
    - `DW_FORM_string`
    - ...
  - Hodnota

DIE
type = compile_unit
comp_dir (str) = /tmp/test
name (str) = test.c
producer (str) = GCC

Hlavní DIE

```
DW_TAG_compile_unit
    DW_AT_producer          GNU C 4.7.2 20130108 [gcc-4_7-branch revision 195012]
    DW_AT_language          DW_LANG_C89
    DW_AT_name              test.c
    DW_AT_comp_dir          /tmp/test
    DW_AT_low_pc            0x00000000
    DW_AT_entry_pc          0x00000000
    DW_AT_stmt_list         0x00000000
 ...
DW_TAG_subprogram
    DW_AT_external          yes(1)
    DW_AT_name              main
    DW_AT_decl_file         0x00000001 /tmp/test/test.c
    DW_AT_decl_line         0x00000019
    DW_AT_prototyped        yes(1)
    DW_AT_type              <0x0000005b>
    DW_AT_low_pc            0x00000000
    DW_AT_high_pc           0x0000011d
    DW_AT_frame_base         ...
    DW_AT_GNU_all_call_sites yes(1)
    DW_AT_sibling           <0x00000770>
```

Atributy DIE

Iterace přes všechny atributy v jednom DIE

- `ptrdiff_t dwarf_getattrs(...)`
  - `Dwarf_Die *die`: z `dwfl_nextcu`
  - `int (*callback)(Dwarf_Attribute *, void *)`: vaše funkce, která se zavolá pro každý atribut v DIE
  - `void *arg`: něco, co se předá do `callback` jako 2. parametr
  - `ptrdiff_t offset`: `0`

Operace nad atributem

- O co jde: `dwarf_whatattr(Dwarf_Attribute *attr)`
  - Vrací `DW_AT_*` (`DW_AT_name` apod.)
- Typ obsahu: `dwarf_whatform(Dwarf_Attribute *attr)`
  - Vrací `DW_FORM_*` (`DW_FORM_string` apod.)
- Obsah: `dwarf_form*(Dwarf_Attribute *attr)`
  - Např. `dwarf_formstring`

Atributy DIE – příklad

```
static int dump_die_attr(Dwarf_Attribute *attr, void *ptr)
{
   printf ("AT=%x FORM=%x\n", dwarf_whatattr(attr), dwarf_whatform(attr));
   if (dwarf_whatform(attr) == DW_FORM_strp)
      printf ("    name=%s\n", dwarf_formstring(attr));
   return DWARF_CB_OK;
}

static void dump_dwfl(Dwfl *dwfl)
{
   Dwarf_Die *die = NULL;
   Dwarf_Addr bias;

   while ((die = dwfl_nextcu(dwfl, die, &bias))) {
      dwarf_getattrs(die, dump_die_attr, NULL, 0);
}
```

Úkol

Výpis atributů

1. Definujte funkci (háček) pro `dwarf_getattrs`
2. V ní vypište informace
   - O jaký atribut jde (`dwarf_whatattr(attr)`)
   - Typ obsahu atributu (`dwarf_whatform(attr)`)
   - Adresu, pokud se jedná o typ `DW_AT_low_pc`
   - Řetězec a typ atributu `DW_AT_*`, pokud se jedná o typ hodnoty `DW_FORM_string` nebo `DW_FORM_strp`
3. V těle cyklu `dwfl_nextcu` zavolejte `dwarf_getattrs`
4. Přeložte a spusťte

Průchod skoro-stromem

Procházení okolních DIE

- `int dwarf_child(Dwarf_Die *die, Dwarf_Die *result)`
  - Do `result` vloží potomka `die`
  - Vrací 0 (OK), 1 (žádný potomek), -1 (chyba)
  - `result` musí být ukazatel na lokální proměnnou
- `int dwarf_siblingof(Dwarf_Die *die, Dwarf_Die *result)`
  - Do `result` vloží sourozence `die`
  - Vrací 0 (OK), 1 (žádný sourozenec), -1 (chyba)

Úkol

Výpis potomků CU

1. Vezměte 1. potomka od CU (`dwarf_child`)
2. Iterujte přes sourozence potomka (`dwarf_siblingof`)
3. Pro všechny potomky vypište typ DIE (`dwarf_tag`)
   - Nejprve zjistěte, které tam jsou
   - Potom vypisujte v textové podobě
4. Rozšiřte na celý podstrom, nejen první úroveň
5. Přeložte a spusťte