

PB173 – Binární programování Linux

VI. DWARF

Jiri Slaby

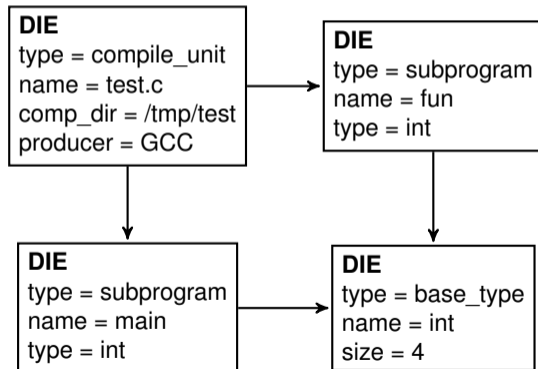
Fakulta informatiky
Masarykova univerzita

25. 10. 2016

- Strukturované ladicí informace
 - Zápis komplikovaných výrazů
 - `void klist_init(struct klist *k, void (*get)(struct klist_node *), void (*put)(struct klist_node *));`
 - Vytvářené překladačem
- Nezávislost na jazyku a formátu souborů
- Dokumentace
 - Introduction to the DWARF Debugging Format (10 stran, obrázky)
 - DWARF Debugging Information Format (Version 4) (> 300 stran)

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 je v ELFu uložený v `.debug_*` sekcích

- `.debug_info`: všechny DIE z lesa stromů
- `.debug_abbrev`
 - Typ položek v `.debug_info`
 - Figure 48 v DWARF4
- `.debug_line`: překlad instrukce ↔ řádek kódu
- `.debug_loc`: popis maker
- `.debug_str`: řetězce odkazované z `.debug_info`
- Výpis
 - `readelf --debug-dump`
 - `objdump --dwarf`
 - `dwarfdump` (z `libdwarf`)

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*

- Knihovny jsou součástí elfutils
- Lze je používat střídavě (podobně 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)`
- Chyba na řetězec: `const char *dwfl_errmsg(int err)`
 - `err: -1` pro poslední
- Dokumentace
 - V hlavičkových souborech
- Knihovny při překladu: `gcc ... -ldw`

Inicializace libdwfl

- 1 Nainstalujte si libdw (a libdwfl)
- 2 Vytvořte si main
- 3 Definujte si háčky
 - Statická 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

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))) { ... }
```


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

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>
```

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_getattns(die, dump_die_attr, NULL, 0);
    }
```

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

Procházení okolních DIE

- `int dwarf_child(Dwarf_Die *die, Dwarf_Die *result)`
 - Do `result` vloží 1. 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)

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