VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

GRAFICKÝ PROHLÍŽEČ A JEDNODUCHÝ EDITOR ELF BINÁRNÍHO SOUBORU

THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **TOMÁŠ VEŠKRNA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19922/2016/xveskr04

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Veškrna Tomáš**
Obor: Informační technologie
Téma: **Grafický prohlížeč a jednoduchý editor ELF souboru**
Graphical Viewer and Simple Editor of ELF File
Kategorie: Překladače

Pokyny:

1. Seznamte se s multiplatformním formátem pro binární soubory ELF - Executable and Linkable Format. Seznamte se s knihovnami pro čtení a manipulace s ELF.
2. Dle pokynů vedoucího vyberte knihovnu pro práci s ELF a analyzujte její schopnosti ohledně práce s netradičními úpravami ELF souborů.
3. Podle pokynů vedoucího navrhněte způsob zobrazení zadaného ELF souboru z různých pohledů včetně interaktivní navigace odpovídajícími částmi těchto pohledů. Podporujte i editaci ELF souboru a schopnost se vypořádat s ručně upraveným ELF souborem.
4. Navrženou aplikaci implementujte jako multiplatformní (Linux a další vybraná platforma).
5. Vyhodnoťte vytvořenou aplikaci a navrhněte její další vylepšení.

Literatura:

- TOOL INTERFACE STANDARD (TIS). *Executable and Linking Format (ELF) Specification* [online]. Version 1.2. May 1995 [cit. 2014-04-08]. Dostupné na https://refspecs.linuxbase.org/elf/elf.pdf.
- M. Wilding, D. Behman: Self-Service Linux(R): Mastering the Art of Problem Determination. Prentice Hall, 2005.
- Dle pokynů vedoucího a konzultanta

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Křivka Zbyněk, Ing., Ph.D.**, UIFS FIT VUT
Konzultant: Hack Robin, RedHatCZ
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zabývá tvorbou jednoduchého grafického prohlížeče a editoru ELF binárních souborů. Cílem práce je poskytnout jednodušší práci s ELF soubory, i když byly dříve poškozeny špatnou úpravou. Výsledný software musí být multiplatformní, tudíž pro zpracování grafického uživatelského rozhraní je použita knihovna Qt. Pro práci s ELF soubory je použita knihovna elf.h. Samotná aplikace je implementována pomocí jazyka C++. Výslednou aplikaci je možné spustit na operačním systému Linux, potažmo Microsoft Windows.

Abstract

This thesis specify, design and develop graphial viewer and editor of ELF binary files. The goal of this thesis is to provide easier manipulation with ELF files, although they were corrupted by wrong editing. The final software must be multiplatform, so Qt library is used for the implementation of its graphical user interface. For manipulation with ELF files is uses elf.h library. The application is implemented in C++. The realized tool can be used on both Linux operation system and Microsoft Windows.

Klíčová slova

ELF, Qt, C++, ELFIO, grafický editor

Keywords

ELF, Qt, C++, ELFIO, graphical editor

Citace

VEŠKRNA, Tomáš. *Grafický prohlížeč a jednoduchý editor ELF binárního souboru.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Křivka Zbyněk.

Grafický prohlížeč a jednoduchý editor ELF binárního souboru

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky, Ph.D. Další informace mi poskytl pan Robin Hack. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Veškrna
23. května 2017

Poděkování

Chtěl bych poděkovat Ing. Zbyňku Křivkovi, Ph.D. za vedení mojí bakalařské práce a panu Robinovi Hackovi za odbornou pomoc při psaní této práce.

Obsah

Kapitola 1

Úvod

Formát ELF je velmi rozšířený standard. V roce 1999 byl zvolen pro Unixové systémy jako standardní formát binárních souborů, tedy je určen převážně pro Unixové systémy. ELF formát je velmi flexibilní. Není vázán na konkrétní procesor ani konkrétní architekturu, a právě proto je dnes používán i na některých ne-Unixových a mobilních systémech, dokonce na některých herních konzolích.

Přesto, že je dnes ELF formát velmi rozšířený, tak existuje jen velmi málo editorů binárních souborů tohoto formátu. V tomto malém počtu editorů však neexistuje žádný s grafickým uživatelským rozhraním, který by zobrazil například graf sekcí, což editor, o kterém pojednává tato práce, umí.

Čtenáři se v této práci mohou dozvědět, jak vypadá a funguje formát ELF, jaká je jeho struktura, jak je soubor vytvořen a co je k tomu potřeba, aby byl výsledný soubor ve formátu ELF. V práci je objasněn i postup návrhu a implementace výsledné aplikace. Tato práce může sloužit jako manuál výsledné aplikace.

V následující části 2 budou popsány požadavky na implementaci praktické části práce. Kromě analýzy požadavků bude tato kapitola obsahovat rozbor existujících řešení.

Další kapitola 3 se bude zabývat ELF formátem samotným a proč je tak rozšířený.

Po rozboru formátu ELF budou analyzovány použité knihovny 4 pro práci s ELF formátem, pro implementaci grafické aplikace a pro překlad strojového kódu do jazyka symbolických adres. Návrh aplikace bude popsán v kapitole 5. Tato kapitola se dále zabývá použitím konkrétních knihoven, grafickým vzhledem aplikace, implementací aplikace, editace a zpracování porušených souborů a testování výsledné aplikace. Poslední kapitolou je závěr 6, který shrnuje body zadání, do jaké míry se je povedlo splnit a navrhuje vylepšení výsledné aplikace.

Kapitola 2

Analýza

Tato kapitola se bude zabývat rozborem požadavků kladených na tuto bakalářkou práci. Nejprve je provedena analýza zadání a poté je proveden krátký průzkum existujících řešení.

2.1 Analýza zadání

Základním požadavkem pro implementaci je použití programovacího jazyka C/C++. Všechna další rozhodnutí, jako jsou použité knihovny a způsoby implementace jsou odvíjeny právě od tohoto požadavku.

Nejdůležitějším požadavkem a vlastně náplní celé této práce je zobrazit obsah binárního ELF souboru pomocí aplikace s grafickým uživatelským rozhraním (dále jako GUI) a implementovat aplikaci tak, aby si uměla poradit s porušenými ELF binárními soubory a podporovala jejich editaci.

Zobrazení obsahu ELF souboru pomocí hexadecimálních znaků není žádnou odlišností od jiných existujících nástrojů a celkově je zbytečné k těmto účelům používat aplikaci s GUI. Mezi hlavní požadavky patří implementace grafu sekcí. Binární ELF soubory jsou strukturovány do jednotlivých bloků. Graf sekcí bude tyto bloky reprezentovat a pomocí tohoto grafu je možno zobrazit vlastnosti a popřípadě obsah jednotlivých bloků.

Velmi důležitou vlastností výsledného softwaru je, aby si dokázal poradit s porušenými soubory, které vznikly neoprávněným zásahem do jejich struktury, který poruší pravidla konzistence souboru. Těmito neoprávněnými zásahy mohou být například: chybný překlad zdrojového kódu do formátu ELF nebo manuální poškození souboru pomocí editoru binárních souborů.

Jedním z bodů zadání je i podporování editace ELF binárních souborů. Tuto vlastnost ocení uživatel v případě opravování poškozených ELF binárních souborů.

Cílovým operačním systémem je Linux. Implementace aplikace pro ostatní operační systémy (dále jen jako OS) jako jsou Mac OS X a Microsoft Windows, jsou vhodnými doplňky.

Posledním bodem zadání je vyhodnotit výslednou aplikaci a navrhnout její vylepšení. O tomto tématu pojednává kapitola Závěr 6.

2.2 Existující řešení

Jak bylo zmíněno v úvodní kapitole 1, neexistuje moc aplikací určených pro práci s ELF binárními soubory. Jako jednoduché náhražky mohou sloužit editory, jež převádí hexadeci-

mální znaky na znaky abecedy a tím je možné upravovat tyto soubory na nejnižší úrovni. Dále existují utility pro práci s ELF binárními soubory. Tyto utility jsou však spíše informativní, kdy vypíší klíčové vlastnosti daného souboru a zbylým obsahem se příliš nezabývají. V této kapitole se zaměříme na existující aplikace a utility pro práci s ELF binárními soubory.

2.2.1 HT Editor

HT Editor je software [15], který se asi nejvíce podobá výslednému softwaru, který je v této práci popisován. Hlavní rozdíl spočívá v tom, že tato aplikace není implementována graficky, spouští se z okna terminálu, ve kterém i aplikace běží. na podobném principu fungují třeba textové editory jako třeba vi, vim, emacs atd. HT editor je schopný zobrazit čistou podobu binárního souboru tak, že převede hexadecimální znaky na znaky ASCII. Kromě tohoto pohledu umí zobrazit i jednotlivě části jako jsou: vlastnosti souboru, hlavičku souboru, strojový kód, hlavičky sekcí a jiné. Ovšem neumí zobrazit jednotlivé sekce.

Jelikož je tento program spouštěn v terminálu, je jeho ovládání pro nezkušeného uživatele velmi obtížné. Zdrojový kód je zobrazen jako jeden celek a i pro velmi jednoduché programy je odpovídající strojový kód velmi obsáhlý a špatně se v něm orientuje.

Program je již velmi zastaralý a jeho vývoj je pozastaven. Orientace v něm je ze začátku trochu těžkopádná, ale po chvíli práce se ukázalo, že tento program je velmi užitečný a poslouží jako dobrý vzor pro implementaci této práce.

2.2.2 Utilita readelf

Jak je zmíněno v úvodu této sekce, utility pro práci s ELF soubory jsou spíše brány jako nástroj informativní a tato utilita [13] není výjimkou. Dokáže vypsat ELF hlavičku, přehled všech sekcí, segmentů a jiné informativní bloky. Obsahy jednotlivých segmentů a sekcí tato utilita nezpracovává. Utilita readelf je velmi inspirativní a strukturu binárního souboru zobrazuje přehledně a správně. Proto je vhodným nástrojem pro kontrolu údajů o binárních souborech, které bude zobrazovat výsledná aplikace.

2.2.3 Utilita elfdump

Tato utilita [2] naopak pracuje se sekcemi. Nedokáže zobrazit hlavní informace, které lze přečíst z hlavičky souboru, zato dokáže zobrazit vlastnosti jednotlivých sekcí, ale bohužel už ne jejich obsah.

2.2.4 Utilita scanelf

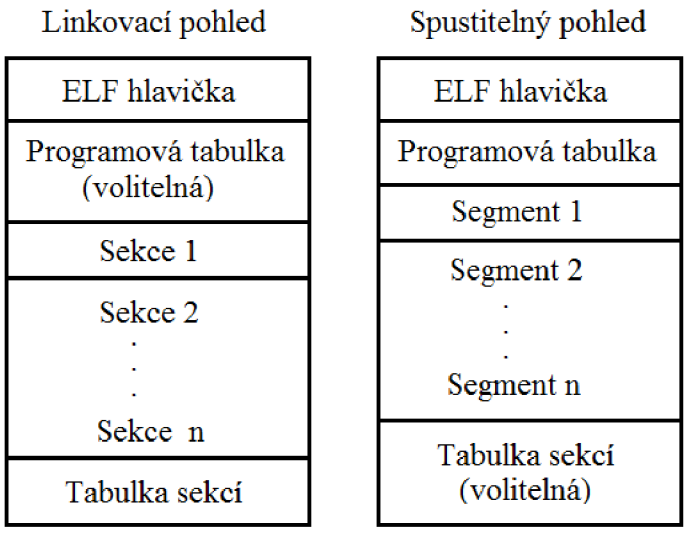
Utilita *scanelf* [5] je ze všech zmíněných asi nejstručnější. Je schopna zobrazit opravdu jen klíčové vlastnosti ELF souboru, jako jsou: typ souboru, formát uložených dat, velikost souboru v bytech a spoustu jiných klíčových informací.

Kapitola 3

Formát ELF

Všechny informace jsou čerpány z originální dokumentace pro ELF formát [14].
Existují 3 hlavní typy objektových souborů ELF:

- Realokovatelný soubor – Obsahuje kód a data pro slinkování s jiným objektovým souborem a vytvoření spustitelného souboru nebo sdíleného souboru.
- Spustitelný soubor – Obsahuje program pro vykonání.
- Sdílený objektový soubor – Obsahuje kód a data pro slinkování do dvou kontextů.
  1. Linker může slinkovat daný soubor s jiným realokovatelným či sdíleným objektovým souborem a vytvořit tak nový sdílený objektový soubor.
  2. Linker může slinkovat daný soubor s jiným spustitelným či sdíleným objektovým souborem a vytvořit tak nový spustitelný nebo sdílený objektový soubor.



Obrázek 3.1: Formáty objektových souborů [14]

Linkovací pohled popisuje ELF soubor, který je možno slinkovat a vytvořit spustitelný soubor. Tento pohled obsahuje ELF hlavičku, která je vždy na začátku, pořadí ostatních

částí je volitelné. ELF hlavička obsahuje informace o daném souboru a odkazuje na povinnou tabulku sekcí. Tabulka sekcí ukazuje na jednotlivé sekce v souboru. Kromě těchto částí obsahuje soubor sekce, kdy každá sekce obsahuje informace jako: instrukce, data, tabulku symbolů, informace o realokaci, tabulku řetězců, textové informace o souboru. Programová tabulka je volitelná.

Spustitelný pohled popisuje spustitelný soubor. na začátku je opět hlavička ELF, která má stejné vlastnosti jako v linkovém pohledu. Povinnou sekcí je programová tabulka, která ukazuje na segmenty v souboru. Segmenty obsahují libovolný počet sekcí. Tyto sekce jsou sdružovány do jednotlivých segmentů na základě podobných vlastností. Sekce v tomto pohledu mají opět stejné vlastnosti jako u linkovacího pohledu. Tabulka sekcí je zde volitelná.

Nyní budou detailněji popsány nejdůležitější části ELF formátu.

3.1 Hlavička ELF

Hlavička ELF je částí nejdůležitější. Má formát struktury a uchovává důležité informace o souboru. Zejména příznaky souboru, které označují formát ELF, typ souboru, cílovou platformu a kódování dat. Dále offsety na tabulku sekcí nebo programovou hlavičku, počty sekcí a segmentů, velikosti jednotlivých položek v tabulce sekcí a programové tabulce. Nakonec obsahuje index do tabulky sekcí, který říká, která sekce obsahuje jména jednotlivých sekcí. ELF hlavička je vždy na začátku souboru, viz obr. 3.1. Z toho vyplývá, že ostatní bloky nemají určité pořadí, a proto se v souboru nachází na libovolném místě. K lokalizaci jednotlivých segmentů slouží programová hlavička. Podobně je na tom hlavička sekcí, která odkazuje na jednotlivé sekce. Aby bylo možné k hlavičce sekcí nebo k programové hlavičce přistoupit, je v ELF hlavičce vždy offset pro daný blok. Tento offset udává vzdálenost v bytech mezi začátkem souboru a začátkem příslušné hlavičky. Datové struktury jako jsou: hlavička ELF, hlavička sekcí a programová hlavička mají vždy konstantní velikost. Mohou se lišit pouze v případě rozličných architektur, na kterých byly výsledné soubory přeloženy (32/64 bit), takže není nutné si pamatovat velikost dané hlavičky nebo značit její konec. V dokumentaci ELF [14] není aktuální přehled hodnot reprezentující architektury a OS, na kterých byl ELF binární soubor vytvořen. Byl použit externí zdroj, odkud byly tyto hodnoty získány [16].

3.2 Hlavička sekcí

Hlavička ELF obsahuje odkaz na hlavičku sekcí a počet sekcí v souboru. Program, který pracuje s ELF souborem tedy ví, kolik sekcí soubor obsahuje a kde je najít. Hlavička sekcí má formát pole struktur. Tyto struktury jsou v souboru uloženy za sebou. Toto pole začíná na offsetu, který je uložený v hlavičce souboru. Díky počtu sekcí, který je taktéž uložen v hlavičce, ví program, který čte daný ELF soubor, kolik struktur za sebou následuje.

Každá struktura v hlavičce sekcí obsahuje jméno sekce, typ, příznaky, velikost, offset, podle kterého lze obsah sekce najít, zarovnání a velikost jednotlivých položek. Obsah sekce je určený podle typu a příznaků. Tyto typy budou popsány níže 3.4. Záznam v hlavičce sekcí může být prázdný, nemusí odkazovat na sekci.

3.3 Programová hlavička

Programová hlavička je uložena v souboru podobně jako hlavička sekcí. Liší se jen záznamy v ní.

Každý záznam obsahuje typ segmentu, offset, fyzickou a virtuální adresu, které udávají kde začíná daný segment, velikost segmentu v souboru a v paměti, příznaky a přiřazení. Virtuální adresa je adresa, kam se při vytváření spustitelného souboru segment přesune a utvoří tak strukturu pro spustitelný soubor. Virtuální velikost segmentu v paměti pak udává, jak se velikost segmentu změní při načtení a spuštění souboru. Tato změna velikosti segmentu nastává v případě alokace potřebného místa pro činnost programu. Alokované místo navíc je vyplněno samými nulami. Záznam v programové hlavičce může být prázdný, tedy nemusí ukazovat do paměti souboru.

V dokumentaci k ELF formátu [14] se v sekci věnované programové hlavičce vyskytuje malá, ale zásadní dezinformace. Liší se pořadí proměnných ve strukturách pro 32-bitové a 64-bitové systémy. Naštěstí tento údaj byl správně uveden v knihovně *elf.h* [4]. Porovnání můžete vidět na obrázku 3.2, kdy se liší položky struktury pro 64-bitové systémy, což se u ostatních struktur neděje. Struktura vlevo je vyjmuta z ELF dokumentace a zbylé jsou vyjmuty z knihovny *elf.h*.

```
typedef struct {
        Elf32_Word      p_type;
        Elf32_Off       p_offset
        Elf32_Addr      p_vaddr;
        Elf32_Addr      p_paddr;
        Elf32_Word      p_filesz
        Elf32_Word      p_memsz;
        Elf32_Word      p_flags;
        Elf32_Word      p_align;
} Elf32_Phdr;
```

```
typedef struct elf32_phdr{
  Elf32_Word    p_type;
  Elf32_Off     p_offset;
  Elf32_Addr    p_vaddr;
  Elf32_Addr    p_paddr;
  Elf32_Word    p_filesz;
  Elf32_Word    p_memsz;
  Elf32_Word    p_flags;
  Elf32_Word    p_align;
} Elf32_Phdr;
```

```
typedef struct elf64_phdr {
  Elf64_Word p_type;
  Elf64_Word p_flags;
  Elf64_Off p_offset;
  Elf64_Addr p_vaddr;
  Elf64_Addr p_paddr;
  Elf64_Xword p_filesz;
  Elf64_Xword p_memsz;
  Elf64_Xword p_align;
} Elf64_Phdr;
```

Obrázek 3.2: Porovnání programových hlaviček z dokumentace [14] a z knihovny elf.h [4]

3.4 Sekce

Na obrázku 3.1 můžete vidět, jak zhruba vypadá struktura ELF souboru. Tento obrázek může navozovat mylný dojem, že sekce obsahuje všechny informace o ní samotné v téže bloku. Pravda je, že všechny informace o sekci jsou uvedeny v příslušném záznamu hlavičky sekcí a v bloku sekce už jsou uložena pouze data. Sekce mohou být sdružovány do segmentů. Sekce může být prázdná, takže hlavička dané sekce neukazuje do paměti souboru a velikost dat sekce je nulová. Sekce se nemohou v žádném případě překrývat. Typ a formát dat udává typ sekce. Zde jsou stručně popsány typy sekcí a jejich číselné hodnoty:

- SHT_NULL (0) – Záznam v tabulce sekcí je prázdný a obsah sekcí je nulový.
- SHT_PROGBITS (1) – Obsah této sekce není nijak standardizovaný, formát a význam této sekce je specifikovaný programem.
- SHT_SYMTAB (2) – Sekce reprezentuje tabulku symbolů.

- SHT_STRTAB (3) – Sekce reprezentuje tabulku řetězců, která obsahuje řetězce znaků pro celý soubor.
- SHT_RELA (4) – Sekce obsahuje záznamy o realokaci sekcí při kompilaci.
- SHT_HASH (5) – Sekce obsahuje hašovací tabulku.
- SHT_DYNAMIC (6) – Sekce obsahuje informace pro dynamické slinkování.
- SHT_NOTE (7) – Sekce obsahuje informace, které označují soubor pro lepší kompatibilitu.
- SHT_NOBITS (8) – Sekce je uložena někde v souboru, ale její obsah je nulový.
- SHT_REL (9) – Sekce obsahuje realokovatelné položky.
- SHT_SHLIB (10) – Tento typ sekce nemá specifikovaný význam.
- SHT_DYNSYM (11) – Sekce reprezentuje dynamickou tabulku symbolů.

K upřesnění významu slouží ještě příznaky, které specifikují, jak s obsahem dané sekce program naloží. Existují 3 základní příznaky:

- SHF_WRITE (1) – Sekce obsahuje data pro zápis během spuštění programu.
- SHF_ALLOC (2) – Sekce zabírá místo v paměti během spuštění programu.
- SHF_EXECINSTR (4) – Sekce obsahuje spustitelné instrukce.

3.5 Speciální sekce

Sekce 3.4 mají sémantiku a obsah určený podle typu sekce a příznaků. Existují ale předdefinované sekce, které mají konkrétní význam. Zde jsou stručně popsány ty nejdůležitější:

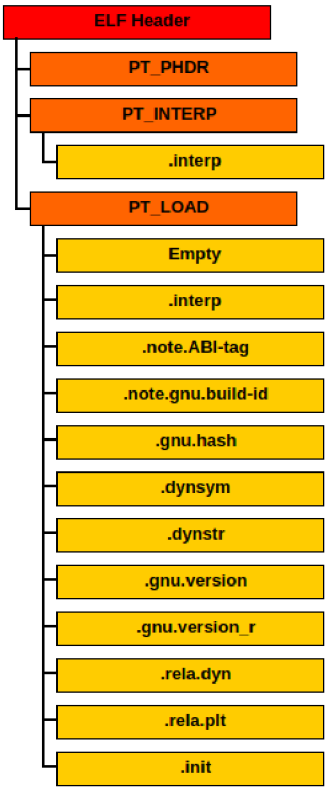
- .bss – Sekce obsahuje údaje o velikosti a umístění sekce, která slouží pro umístění staticky alokované proměnné. Program při spuštění vytvoří tuto sekci, naplní ji proměnnými a inicializuje je na nulovou hodnotu.
- .data – Sekce obsahuje inicializované data.
- .fini – Sekce obsahuje instrukce, které se provedou, když program skončí normálně.
- .got – Sekce obsahuje tabulku globálních offsetů. Před realokací jednotlivých sekcí spočítá dynamický linker absolutní adresy symbolů a vloží je do tabulky globálních offsetů.
- .hash – Sekce reprezentuje hašovací tabulku. Hašovací tabulka obsahuje odkazy do tabulky symbolů na základě jejich jmen. Sestává se z dvou proměnných a dvou polí. Tyto proměnné obsahují velikost daných polí. První pole obsahuje položky pro každé jméno, které bylo do hašovací tabulky vloženo. Pokud by však bylo do hašovací tabulky vloženo jedno jméno vícekrát, je k dispozici druhé pole, které řetězí položky se stejnými jmény.

- .init – Sekce obsahuje instrukce, které se provedou, ještě před voláním hlavního programu. V případě programovacího jazyka C, před voláním funkce main.
- .interp – Sekce obsahuje cestu k překladači programu. Systém při čtení ELF souboru přečte segment určený pro interpret. Obdrží cestu ke konkrétnímu interpretu a vytvoří pro něj počáteční spustitelný obraz ze segmentu pro překladač. Další činnost už záleží na interpretu.
- .plt – Sekce obsahuje tabulku procedurálních propojení. Podobně jako u tabulky globálních offsetů, tak tato sekce obsahuje absolutní adresy pro volání mezi funkcemi, dokonce i mezi různými ELF soubory.
- .rel*name* – Sekce obsahuje informace o realokaci určité sekce. Realokace je nutná, pokud se v některé ze sekcí vyskytne symbolická reference na její symbolickou definici. Například, když program volá funkci, přidružené instrukce volání musí přenést kontrolu na patřičnou cílovou adresu k vykonání. *Name* je proměnná, která ukazuje na sekci, kterou bude třeba realokovat. Název sekce může vypadat takto: *.rel.text* .
- .rodata – Sekce obsahuje data pouze pro čtení.
- .shstrtab – Sekce obsahuje jména sekcí. Jména sekcí jsou uložena v tabulce řetězců.
- .strtab – Sekce obsahuje tabulku řetězců. Tato tabulka obsahuje řetězce pro celý ELF soubor. Řetězce jsou mezi sebou odděleny nulovými znaky. Do tabulky řetězců je možné se odkazovat pomocí indexů. Program tak čte jednotlivé znaky, dokud nenarazí na nulový znak. Díky tomuto je možné šetřit místo a odkazovat se na podřetězce, takže jedna položka může být použita vícekrát. V tabulce řetězců se mohou objevit i prázdné řetězce.
- .symtab – Sekce reprezentuje tabulku symbolů. Tabulka symbolů obsahuje informace potřebné k nalezení a realokaci symbolických definicí a referencí. Je ve formátu pole struktur. Pro realokovatelné soubory obsahují jednotlivé položky v tabulce symbolů offsety k daným sekcím. ve sdílených a spustitelných souborech obsahují virtuální adresy do paměti.
- .text – Sekce obsahuje instrukce programu.

Toto byl stručný popis typů sekcí. Pro kompletní informace ohledně formátu ELF si přečtěte originální dokumentaci v angličtině [14].

3.6 Segmenty

Segment sdružuje sekce s podobnými vlastnostmi, jako sekce určené pro interpret, data, poznámky, komentáře, dynamické slinkování a další. Segment může obsahovat jednu a více sekcí, ale může nastat situace, že segment bude zcela prázdný, jak můžete vidět na obrázku níže 3.3. Segmenty se mohou překrývat, tedy jednotlivé sekce mohou být obsaženy ve více segmentech.



Obrázek 3.3: Reprezentace segmentů v ELF souboru

Kapitola 4

Přehled knihoven

V této kapitole jsou představeny použité knihovny pro práci s ELF soubory a pro vypracování GUI. Pro srovnání jsou uvedeny i konkurenční knihovny a důvody, proč nebyly vybrány právě ony.

4.1 Knihovny pro vypracování GUI

Jeden z požadavků kladených na výslednou aplikaci 2.1 byl, že aplikace musí být multiplatformní. Operační systémy, které jsou podporovány: Linux a Microsoft Windows. Pro případnou budoucí implementaci na Mac OS X byly hledány knihovny, které podporují všechny tři OS. Pro srovnání byly vybrány 3 knihovny, které tyto požadavky splňují: wxWidget, CEGUI a Qt.

4.1.1 wxWidget

Framework wxWdget [9] je vyvíjena pro řadu OS, i těch, které musí výsledný software podporovat. Tento framework podporuje práci s mnoha programovacími jazyky, včetně C++. Mezi aplikace založené na wxWidget patří například: Audacity, Code::Blocks a jiné. Framework wxWidget nemá tak velkou četnost tutoriálů jako Qt, a proto není vybrán pro implementaci této aplikace.

4.1.2 CEGUI

CEGUI [12] je multiplatformní framework. Jeho hlavní nevýhodou a příčinou, proč tento framework nebyl použit pro tento projekt je, že je primárně určen pro tvorbu video her a obecně pro renderování scény, což v tomto projektu není využito. I přesto by se dal použít pro tvorbu aplikačního GUI, ale rozšíření tohoto frameworku není tak rozsáhlé jako u Qt frameworku.

4.1.3 Qt

Qt [6] je framework určený primárně pro jazyk C++, ale existuje i pro ostatní jazyky, například: Python, Java, C# atd. Je multiplatformní a je primárně určen pro tvorbu GUI pro aplikační software. Qt framework podporuje práci se SQL i XML. Jeho velkou výhodou je, že je velmi rozšířený. Důkazem mohou být projekty jako: Skype, Google Earth, Virtual Box, které jsou implementované právě pomocí Qt. Qt má dokonce svoje vývojářské

prostředí: Qt Creator [7], které umožňuje jednoduchou tvorbu GUI i přeložení a spuštění aplikace. Qt Creator dokonce sestaví Makefile, který dokáže přeložit celý projekt a vytvořit tak výslednou aplikaci. Tudíž není potřeba pro přeložení a spuštění mít nainstalovaný Qt Creator.

Framework Qt je založen na objektově orientovaném programování. Všechny grafické elementy jsou definovány jako objekty. Mají implicitně definované metody a pomocí dědičnosti lze vytvářet své objekty, rozšířené o metody, které objekty od vývojářů neposkytují.

Framework Qt je velmi rozšířený a má širokou uživatelskou komunitu. Existuje na něj spousta návodů a video tutoriálů. A hlavně díky četnosti návodů na použití a povedenému vývojovému prostředí je Qt framework zvolen pro vývoj této aplikace.

4.2 Knihovny pro práci s ELF soubory

Binární ELF soubory mají specifickou vnitřní strukturu. Je možné vytvořit vlastní knihovnu podle dokumentace [14], anebo použít již vytvořené knihovny, které obsahují všechny datové typy a struktury, které se v ELF binárních souborech nacházejí.

4.2.1 elf.h

Knihovna elf.h [4] obsahuje pouze deklarace použitých prvků ve formátu ELF: struktur, datových typů, ze kterých se struktury skládají a definicí konstant, kterých určité proměnné mohou nabývat. Knihovna neobsahuje žádné operace nad těmito strukturami. Tato knihovna slouží pouze jako šablona pro načtení ELF souboru do paměti a následné operace si musí programátor implementovat sám. Z této knihovny pak vycházejí všechny níže zmíněné knihovny.

Velkou výhodou této knihovny je, že pokud nebude ELF soubor validní, lze dobře poznat, který blok je poškozený. Struktury v knihovně elf.h (ELF hlavička, hlavička sekcí, programová hlavička) obsahují vždy offset, který složí jako ukazatel v oblasti souboru na konkrétní blok. Pokud je tento offset nesmyslný (jeho hodnota je větší než velikost souboru v bytech), tak je jasné, že tento offset je nevalidní. Pokud není hned na první pohled jasné, že je offset špatný, dá se jeho validita zjistit tak, že byty, které se na dané adrese nachází, neodpovídají vnitřní stavbě struktury. Drobné narušení, například změna dat nebo příznaků nemusí ELF soubor poškodit tak, aby se nedal správně číst. Takové chyby se projeví až při spuštění, kdy program nebude pracovat správně.

Pro implementaci na OS Microsoft Windows nebude originální knihovna elf.h fungovat, protože typy, ze kterých jsou složeny struktury ELF formátu jsou deklarovány na Unixovém OS a tím pádem je tato knihovna na Microsoft Windows nepoužitelná. Naštěstí existuje i knihovna elf.h pro OS Microsoft Windows [11], takže stačí použít právě tuto knihovnu.

4.2.2 ELFIO

Knihovna ELFIO [3] umí číst i vytvářet binární soubory ELF. Má sice úzké zaměření, jelikož pracuje pouze se soubory ELF formátu, ale vynahrazuje tento fakt tím, že práce s ELF soubory je jednoduchá a poskytuje funkce nad formátem ELF. Může ELF soubory jak číst, tak vytvářet. Manipulace s knihovnou ELFIO je jednoduchá a existuje k ní dokumentace, která popisuje všechny funkce a obsahuje i praktickou ukázku jejich použití. ELFIO je vytvořená již dlouho a je velmi dobře odladěná. Při drobných chybách ve formátu ELF, jako

je změna bytů uvnitř sekce, nemá ELFIO problém s procházením souboru. Bohužel při zásadních poruchách, které způsobí, že budou porušeny odkazy na jednotlivé bloky nebo dané bloky budou posunuty oproti offsetům, které by na ně měly odkazovat, tak knihovna ELFIO nedokáže přečíst takový soubor a tím nesplňuje požadavek ze zadání, kdy si knihovna musí umět poradit s poškozeným souborem. Knihovna ELFIO pouze vyhodnotí soubor jako poškozený, ale nepodává další informace o druhu poškození nebo oblasti poškození.

4.2.3 Radare2

Knihovna Radare2 [8] poskytuje množství struktur pro ELF formát, podobně jako knihovna elf.h. Radare2 není určen primárně pro ELF soubory, kromě nich podporuje práci s velikou škálou binárních souborů. Dokonce pro různé OS. Tato knihovna se velmi podobá knihovně elf.h. Jediný rozdíl je, že elf.h je určena pro práci s konkrétním formátem. Zato Radare2 by bylo dobré použít, pokud je potřeba pracovat s různými typy binárních souborů.

4.3 Knihovny pro překlad strojového kódu do jazyka symbolických adres

Tato sekce popisuje, jaké disassemblerovské knihovny byly zkoumány a jejich porovnání.

4.3.1 Udis86

Udis86 je unixová knihovna pro překlad strojového kódu do jazyka symbolických adres. Knihovna *udis86* šla lehce nainstalovat a její použití je velmi jednoduché. Jediná věc, která byla potřeba udělat navíc, byla přidat příznak `-ludis86` do souboru Makefile pro správný překlad. Jediný problém s knihovnou *udis86* je, že má slabou podporu na OS Microsoft Windows. Proto v případě implementace OS Microsoft Windows při použití knihovny *udis86* není přeložen strojový kód do jazyka symbolických adres.

4.3.2 Capstone

Všestranný disassembler *Capstone* [10] použitelný pro platformy jako jsou Linux, Microsoft Windows a Mac OS X. Je dokonce implementovaný pro více programovacích jazyků jako jsou: C#, C++, Go, Java atd. *Capstone* se ze začátku jevil jako ideální volba pro překlad strojového kódu do jazyka symbolických adres, ale během nasazování knihovny na OS Linux Ubuntu se objevovali drobné problémy. Knihovna nešla nainstalovat pomocí příkazu uvedeného na jejich stránkách. Problém byl vyřešen pomocí instalace ze zdrojových souborů. Další problém nastal, když byla testována funkcionalita knihovny podle zdrojového kódu, opět z oficiálních stránek knihovny *Capstone*, a kód nefungoval. Jelikož byla knihovna *udis86* 4.3.1 používána již před knihovnou *Capstone*, byla pro překlad strojového kódu do jazyka symbolických adres použita právě knihovna *udis86*. Tyto problémy mohou být samozřejmě způsobeny špatně nainstalovanými nástroji na testovacím systému nebo nekompatibilitou verze OS.

Kapitola 5

Návrh a implementace

V předchozí kapitole jsou představeny knihovny, které je možné použít pro tvorbu výsledné aplikace. Tato kapitola se zabývá použitím konkrétních knihoven pro implementaci aplikace, grafickým vzhledem aplikace, implementací aplikace a jak byla aplikace otestována.

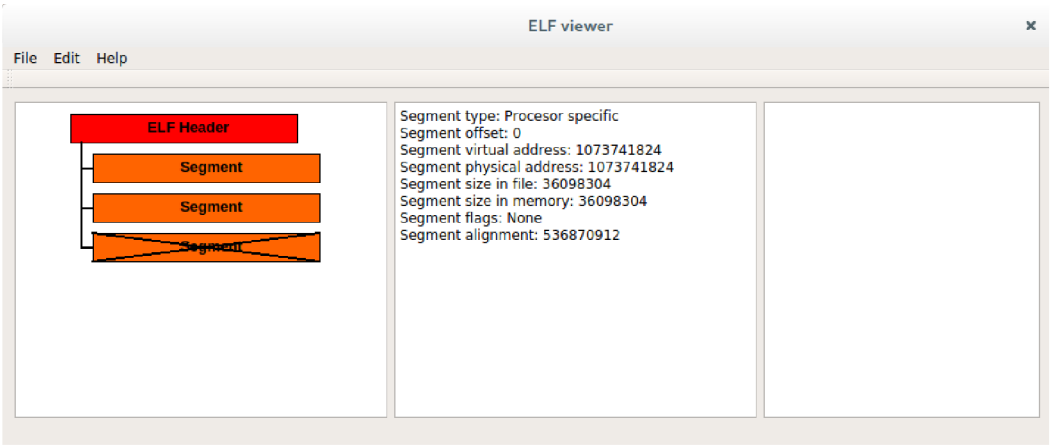
5.1 Použité knihovny

Pro práci s ELF binárními soubory je nejlepší použít knihovnu elf.h. S knihovnou elf.h je nutné pro načítání ELF souborů načítat jednotlivé položky struktur obsažených v souborech. Tuto činnost si musí kontrolovat vývojář sám, takže si může vývojář i sám ohlídat, kdy hodnoty jednotlivých položek neodpovídají dokumentaci [14]. Pro editaci a tím pádem vytváření ELF souboru je knihovna elf.h také dobrou volbou. Stačí si pamatovat změněné části a zbylé části zkopírovat z původního souboru.

GUI je implementováno pomocí frameworku Qt. Při tvorbě GUI byl nejprve použit značkovací jazyk QML. Jazyk QML měl pomoci k lehčí práci s grafickými objekty a softwarovou komunikaci mezi nimi. Zprvu se použití QML jevilo jako dobrý nápad, dokud nepřišlo na řadu zpracování ELF binárních souborů. Bylo obtížné propojit funkcionalitu QML a knihovny z C++ pro zpracování ELF souborů. Při každém volání funkce pro vykreslení grafu v QML se graf vymazal a nebylo možné vykreslovat graf postupně, což by v budoucnu při rozšiřování funkcionality mohlo působit vážné potíže, protože by bylo nutné vykreslování grafu celé přeprogramovat. Proto je pro vypracování GUI vybrán pouze framework Qt bez dalších rozšíření.

5.2 Vzhled aplikace

Pro vypracování tohoto projektu je podmínkou zpracovat ELF soubor pomocí vícero pohledů. Nabízí se hned několik pohledů na ELF binární soubor. Uživatele aplikace zajímá hlavně vnitřní struktura ELF souboru, proto jako jeden z implementovaných pohledů je graf znázorňující vnitřní strukturu konkrétního ELF souboru. Jednotlivé segmenty budou zobrazeny, tak jak budou uloženy v jejich hlavičkách. Pokud však segment bude obsahovat jednu a více sekcí, tyto sekce budou seřazeny a znázorněny jako obsah daného segmentu. Pro ukázku je to vidět na obrázku 3.3. Pokud bude soubor poškozený a aplikace při zobrazování grafu zjistí, která část souboru je poškozená, tak tuto chybu zaznamená do grafu, jako dvakrát přeškrtnutý blok, jak je vidět na obrázku 5.1



Obrázek 5.1: Porušený ELF binární soubor

Soubor zobrazený na obrázku 5.1 je poškozený hned za hlavičkou souboru, kdy byl vložen jeden byte dat navíc. Program rozpoznal chybu až u třetího záznamu v programové hlavičce, takže chyba, která se vyskytne kvůli posunutým offsetům se nemusí projevit hned.

Další informace, které uživatele zajímají, jsou informace o daných blocích uvnitř ELF souboru, tedy přehled atributů konkrétního bloku. Tyto informace budou vypsány do textového okna se jménem konkrétního atributu.

Sekce mají určitý obsah, jak je popsáno v sekci věnované speciálním sekcím 3.5, tento obsah je potřeba zobrazit. Proto je tu třetí pohled, který bude zobrazovat obsah sekcí. Pokud sekce budou obsahovat strojový kód, tak zobrazí jejich překlad do assemblerovských instrukcí.

Pro správu je v horní části okna aplikace menu box [6], pomocí kterého lze otevírat soubory, ukládat soubory, ukončit aplikaci nebo zobrazit nápovědu. Obrázek 5.2 zobrazuje, jak vypadá finální aplikace.

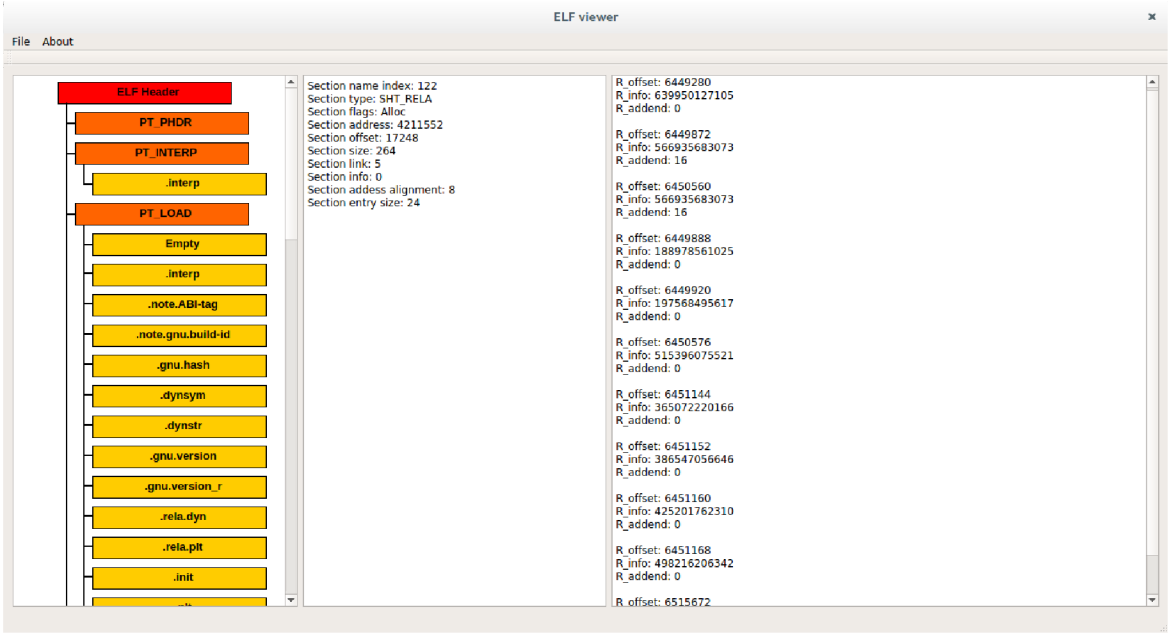
5.3 Tvorba GUI

Tato sekce obsahuje popis, jak je vytvořeno GUI aplikace, jaké objekty byly použity pro tvorbu GUI, porovnání s dalšími použitelnými objekty a proč byly vybrány. Všechny níže zmíněné objekty je možné najít v dokumentaci Qt [6].

5.3.1 Okno aplikace

Základním objektem pro tvorbu GUI ve frameworku Qt je hlavní okno `MainWindow`, které reprezentuje okno aplikace. Bez dalšího programování či konfigurace GUI už má hlavní okno základní vlastnosti: změna rozměrů okna, panel s rozbalovací nabídkou `MenuBar` a tlačítko pro ukončení aplikace. Všechny ostatní objekty je nutno vložit do hlavního okna, takže vzniká jistá hierarchie objektů a pro přístup k danému objektu je nutno projít přes všechny nadřazené objekty.

Rozbalovací nabídka je typicky umístěna v horní části okna aplikace a v případě této aplikace dovoluje uživateli přístup k funkcím jako jsou: správa souborů a manuál na použití aplikace.



Obrázek 5.2: Okno výsledné aplikace

5.3.2 Rozložení objektů

Objekty v Qt lze pozicovat staticky nebo dynamicky. Statické pozicování je standardní pozicování, kdy se danému objektu nastaví výška a šířka a umístí se na určité souřadnice v hlavním okně. Toto pozicování je z hlediska implementace nejjednodušší a hodí se nejvíce pro aplikace s pevně danými rozměry. Kdyby se statické pozicování použilo v dynamickém okně, kdy si uživatel přizpůsobuje rozměry okna aplikace, nastal by problém, že rozměry a tvar okna aplikace by se měnily, ale staticky umístěný objekt by zůstával stále stejný. Vznikalo by nevyužité místo na ploše aplikace, nebo kdyby byl statický objekt větší než okno aplikace, ztrácel by se vizuální kontakt se staticky pozicovaným objektem. Proto je tu dynamické pozicování, které funguje tak, že se danému objektu přiřadí dynamické rozměry. Například výška objektu bude stejná jako výška okna a šířka bude nastavena na polovinu šířky okna aplikace. Tím se docílí efektu, kdy se rozměry objektu budou měnit v závislosti na rozměrech okna aplikace. Potom bude prostor okna aplikace plně využit a dynamicky pozicovaný objekt bude vždy vidět. Za účelem dynamického pozicování existují ve frameworku Qt takzvané `Layouts`, které dokážou objekty skládat horizontálně, vertikálně nebo do mřížky. Tyto `Layouts` lze skládat vedle sebe nebo zanořovat do sebe.

Pro tvorbu zde popisované aplikace bylo použito klasické vertikální rozložení, kdy jednotlivé grafické elementy jsou naskládány vedle sebe, mají výšku nastavenou na výšku okna aplikace a jejich šířka je rovnoměrně rozdělena podle počtu grafických elementů v okně aplikace.

5.3.3 Graf sekcí

Pro graf sekcí lze použít hned několik způsobů, jak ho implementovat:

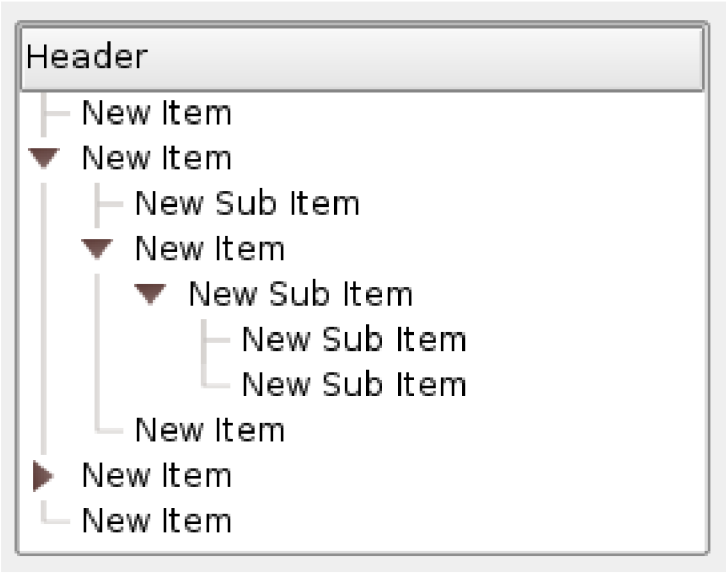
- Implementace grafu pomocí tlačítek – Klasická metoda používaná velmi často, a to nejen ve frameworku Qt. Princip spočívá v generování tlačítek, v případě Qt jsou tlačítka reprezentovány objektem `QPushButton`. Každému tlačítku je přiřazen unikátní

identifikátor. Po zmáčknutí konkrétního tlačítka se zavolá funkce, které se předá jako parametr konkrétní identifikátor. Tato funkce pak vyhodnotí identifikátor z parametrů funkce a vykoná požadovanou činnost, v případě grafu sekcí zobrazí vlastnosti a obsah sekce. Tato metoda není příliš ideální, zvláště, když bude potřeba vygenerovat větší množství tlačítek.

- Implementace grafu pomocí objektu `QTreeView` – Ideální náhradou je objekt `QTreeView`, který je používaný především pro zobrazování adresářové struktury, ale lze ho použít jako graf sekcí, kdy by se sekce zanořovaly do segmentů a tím by se daly minimalizovat a skrýt tak sekce, které uživatele nezajímají. Sekce, které budou zanořeny do segmentů lze minimalizovat a tím snížit rozměry grafu sekcí. Problém nastává, když bude zobrazovaný soubor realokovatelný, bude se tedy skládat pouze ze sekcí. V tomto případě graf sekcí nebude možné nijak minimalizovat. Pro představu, na obrázku 5.3 můžete vidět jak objekt `QTreeView` může vypadat v grafické podobě.

- Implementace grafu pomocí objektu `QGraphicView` – Malou kompenzací může být graf sekcí implementovaný pomocí objektu `QGraphicView`. Tento objekt funguje jako plátno, lze na něj kreslit obrazce a text. Tato metoda dovoluje větší variabilitu, kdy si vývojář může vyvinout graf podle potřeby. Implementace grafu sekcí pomocí této metody je složitější než použití `QTreeView`, ale z dlouhodobého hlediska je lepší použít `QGraphicView`. Například když ELF soubor obsahuje velké množství sekcí, například 100 a více, a žádné segmenty, tak nelze sekce nijak skrýt a graf sekcí bude příliš rozsáhlý. Graf sekcí je implementován právě pomocí objektu `QGraphicView`.

  Při implementaci se vyskytl problém se snímáním polohy myši při kliknutí na graf sekcí. Souřadnice, které byly vráceny pomocí metod snímající akce myši, jako jsou: `mousePressEvent, mouseReleaseEvent, mouseMoveEvent`, nebyly správné. Problém vyřešila deklarace vlastního objektu `MyQGraphicView`, který dědil vlastnosti objektu `QGraphicView`, oproti kterému byl rozšířen o metodu `mousePressEvent`. Po deklaraci nového objektu vracel graf správné souřadnice.



Obrázek 5.3: Objekt `QTreeView` [6]

5.3.4 Vlastnosti a obsah sekcí

Vlastnosti a obsah jednotlivých sekcí lze zobrazit jednoduše jen pomocí textových informací. K tomuto účelu má framework Qt k dispozici objekt `QTextEdit`, který dokáže přidávat a mazat text. Pro zvýrazňování syntaxe je možné použít rozšíření třídy `QSyntaxHighlighter`, které vyznačuje klíčová slova, která vývojář zadá. V tomto projektu jsou použity právě dva objekty typu `QTextEdit`. Jeden pro vlastnosti sekce a druhý pro její obsah. V případě zobrazování obsahu sekcí nastává problém při rozsáhlejších sekcích. Pokud je obsah sekce příliš velký, tak jeho vypsání do objektu `QTextEdit` může trvat déle v závislosti na objemu dat. Například při obsahu sekce, který činí 100 KB může trvat vypsání obsahu několik minut. Konkrétní doba se pak liší podle výkonosti počítače.

5.4 Zpracování binárních souborů ELF

Princip zpracování ELF souboru je jednoduchý. Nejprve je potřeba soubor pro zpracování vybrat. Poté ho zkontrolovat, zda není uvnitř porušen, zaznamenat jeho vnitřní strukturu a promítnout ji do grafu sekcí. Soubor je následně uzavřen. Každá další operace závisí na činnosti uživatele. Pokud bude chtít zobrazit vlastnosti a obsah určité sekce, musí kliknout na odpovídající element v grafu sekcí. Program zaznamená požadavek na obsah sekce. Znovu otevře ELF soubor, načte si konkrétní sekci a zobrazí její obsah do textových oblastí

5.4.1 Výběr souboru

Pro výběr souboru je použita metoda dialogového okna `QFileDialog`. `QFileDialog` je předdefinovaná třída ve frameworku Qt [6]. Po zvolení možnosti „Open File“ se uživateli zobrazí dialogové okno, které mapuje souborovou strukturu sytému, na kterém je aplikace s dialogovým oknem spuštěna. Uživatel najde soubor, který chce načíst, označí ho a potvrdí výběr souboru. `QFileDialog` pak vrátí absolutní cestu ke zvolenému souboru.

5.4.2 Kontrola souboru

Nyní je k dispozici absolutní cesta k souboru, který chce uživatel otevřít. Soubor je otevřen pomocí konstruktoru třídy `fstream` [1]. Je důležité otevírat soubor jako binární soubor a nikoliv jako soubor textový. Načtená data by pak nebyla validní.

Čtení dat ze souboru je implementováno jako čtení jednotlivých struktur, ze kterých je soubor složen. Položky ve strukturách jsou načítány postupně. Když jsou všechny položky struktury načteny, je načase kontrolovat správnost hodnot uvnitř struktury. Hlavními hodnotami, které jsou kontrolovány, jsou offsety v hlavičkách. Nesmí přesahovat rozsah aktuálního souboru nebo se překrývat. Pokud je nalezena špatná hodnota, je tento fakt zaznačen do grafu sekcí a načítání souboru je ukončeno. Při úspěšném zpracování se v grafu sekcí zobrazí vnitřní struktura grafu a soubor je následně uzavřen pro čtení.

Při této kontrole se v paměti programu zároveň vytváří jednosměrně vázaný seznam struktur, který reprezentuje data zobrazená v grafu sekcí. Položky ve struktuře seznamu jsou: jméno, offset, velikost, typ záznamu, ukazatel na následující strukturu. Při startu programu je seznam vytvořen a ukazatel na něj je uložen do proměnné. ELF hlavička souboru je vždy na prvním místě. Pokud je hlavička souboru porušena, program ukončí svou činnost. Toto se děje z důvodu, že pokud hlavička není správná, nemusí jít ani o ELF soubor. Pokud by však o ELF soubor šlo a byly by porušeny reference na hlavičku sekcí nebo programovou hlavičku, tak nemá stejně smysl číst ELF soubor, protože načtená data

by byla s největší pravděpodobnosti špatná. Následně jsou načítány záznamy z programové hlavičky a jsou řazeny v pořadí, v jakém jsou uloženy záznamy v programové hlavičce, což nemusí nutně znamenat, že jsou v souboru uloženy za sebou. Offset a velikost jsou přebrány z hlavičky k danému segmentu. Typ může nabývat hodnot z následujících možných hodnot:

- `HEADER` 0,
- `SEGMENT` 1,
- `SECTION` 2.

Po načtení všech segmentů, pokud tedy byly v souboru obsaženy, je na řadě načtení sekcí. Program načte záznam o sekci, prochází záznamy, které reprezentují segmenty a kontroluje offset a velikost s offsetem aktuálně načtené sekce. Podle offsetů zjistí, zda je sekce obsažena v konkrétním segmentu. Pokud ano, program prochází záznamy přiřazené k segmentu a přidá aktuální sekci na konec. Program pokračuje dále, jelikož může sekce být obsažena ve více segmentech. Pokud však segmenty v souboru nejsou obsaženy, tak řadí sekce postupně za sebou, opět v pořadí, v jakém se nachází v hlavičce sekcí.

Takto je vytvořen jednosměrně vázaný seznam, který reprezentuje graf sekcí. Pokaždé, když uživatel klikne na nějaký blok, prochází se tento seznam a podle souřadnic se určí, kolikátý záznam odpovídá elementu v grafu. Program poté otevře soubor a načte si danou strukturu v souboru, podle offsetu, který je uložen v seznamu.

5.5 Zobrazení grafu

Po kontrole souboru je v paměti programu vytvořen jednosměrně vázaný seznam, který reprezentuje vnitřní strukturu ELF binárního souboru. Avšak pro jednotlivé sekce a segmenty nejsou nastaveny jejich jména, jelikož v době čtení segmentů a sekcí není znám offset sekce, která obsahuje jména sekcí. Program pouze ví, který záznam v hlavičce sekcí reprezentuje tabulku řetězců se jmény sekcí. Po kompletním načtení hlaviček je zavolána funkce, která doplní jména sekcí a segmentů do jednosměrně vázaného seznamu. Sekce mají své názvy uloženy v sekci s označením *.shstrtab*. Segmenty názvy nemají, takže jim jsou přiřazena symbolická jména, která reprezentují jejich účel podle jejich typu. Názvy segmentů a jejich význam:

- PT_NULL – Nemají definovaný význam. Jsou prázdné.
- PT_LOAD – Segmenty s klasickým obsahem. Pokud je však výsledná velikost segmentu větší než velikost segmentu před překladem, tak je rozdílové místo v paměti vyplněno nulami.
- PT_DYNAMIC – Segment, který má tento typ, obsahuje sekci *.dynamic* a rozšiřuje údaje o realokovatelných datech pro intrepret.
- PT_INTERP – Informace určené pro interpret. Tento segment je použitelný pouze pro spustitelné typy typy souborů.
- PT_PHDR – Segment obsahuje programovou tabulku, která obsahuje reference na segmenty.
- PT_NOTE – Segment obsahuje „Note sections“, což jsou sekce s poznámkami, které specifikují obsah a zajišťují lepší překlad pro interpret.

Vykreslování grafu je prováděno v cyklu, kdy jsou načítány jednotlivé položky jednosměrného seznamu. Nejprve je zkontrolována validita konkrétního bloku. Pokud je v pořádku, vykreslí se element s textem, který reprezentuje daný blok a doplní se vazby na nadřazené elementy, které jsou reprezentovány čarami vedoucími k elementům hierarchicky nadřazeným, stejně jako je tomu u objektu `QTreeView` 5.3. Pokud je validita porušena, je vyznačen konkrétní element jako porušený (dvakrát přeškrtnutý) a vykreslování grafu je ukončeno.

5.6 Zobrazení vlastností a obsahu sekcí

S plně zobrazeným grafem sekcí může uživatel zobrazovat obsahy jednotlivých sekcí a jejich hlaviček. Zobrazení hlavičky je jednoduché. Program načte strukturu hlavičky a do textového pole vloží dvojice hodnot: jméno proměnné a její hodnotu.

Zobrazení obsahu sekcí je složitější. Segmenty obsahují pouze sekce, takže není nutné zobrazovat obsah segmentů. Obsah sekcí závisí na jejich typu. Sekce s textovým obsahem a sekce s obsahem závislém na interpretu jsou vypsány tak, jak jsou uloženy v souboru, to jest znak po znaku. Ostatní sekce obsahují data uložená ve strukturách. Nejprve je nutné rozpoznat typ sekce a poté načítat obsah sekce do jednotlivých struktur. Při vypisování struktur jsou opět vypisovány dvojice hodnot: název proměnné a její hodnota. U sekcí se strojovým kódem je obsah dané sekce přeložen pomocí disassembleru do jazyka symbolických adres, ten je následně zobrazen do textového pole.

5.7 Editace a porušené soubory

V této sekci je popsán princip, jak je implementována editace ELF binárních souborů pro zde popisovanou aplikaci. Navíc je zde ukázka, jak lze editovat porušené soubory a jaký vliv na ně editace má.

5.7.1 Editace

Editace ELF binárních souborů je složitý proces. Jak bylo popsáno v kapitole 3, ELF soubory jsou složeny z různě na sebe závislých bloků dat. Všechny obsažené záznamy v ELF souborech jsou adresovány pomocí offsetů, takže při porušení offsetů nebo posunutí dat v ELF souborech může dojít k vážnému porušení validity ELF souborů a tím znemožnění čitelnosti souborů pro interpret. V takových případech může být vhodná editace takovýchto souborů řešením. Na druhou stranu, neuvážená editace ELF binárních souborů může být zdrojem poškození ELF souborů, proto je nutné každou změnu ELF binárních souborů zvážit a neměl by se o ni pokoušet uživatel, který si nebude jistý svým konáním. Kromě offsetů mohou být jednotlivé bloky provázány mezi sebou i pomocí jiných závislostí. Například sekce s názvem *.rel* nebo *.rela* obsahují informace o realokaci konkrétních sekcí. Při posunu dat v ELF binárním souboru by potom mohla nastat situace, že pro validitu souboru by bylo nutné načíst sekce s realokovatelnými informacemi a všechny je přesměrovat na správné místa v paměti. Taková činnost je velmi složitá a úsilí potřebné pro její implementaci sahá až za rámec obtížnosti této práce.

Aplikace zatím umožňuje editovat pouze některé vlastnosti pro tyto objekty: ELF hlavička, programová hlavička a hlavička sekcí. Konkrétně: typ souboru, offset programové hlavičky a offset hlavičky sekcí pro hlavičku ELF, offset segmentu pro programovou hlavičku a offset sekce pro hlavičku sekcí. Tyto vlastnosti jsou nejdůležitější pro editaci ELF

binárních souborů. Ostatní údaje nejsou tak důležité, neboť právě offsety jsou ty údaje, které bývají poškozeny a je nutné je opravit.

Aplikace dovoluje editaci pouze jednoho objektu najednou. To znamená, že po úpravě konkrétních vlastností jednoho objektu je potřeba nejprve změny uložit, než uživatel přistoupí k úpravě dalšího objektu. To se děje kvůli tomu, že soubor může změnit svoji vnitřní strukturu a úpravy učiněné v oblasti, kde soubor mění svoji strukturu by nebyly validní. Názorný příklad můžete vidět níže, v sekci 5.7.2.

Po provedení úprav má uživatel několik možností:

- úpravy může ignorovat,
- úpravy může uložit do jiného souboru,
- úpravy může uložit do aktuálního souboru a ten znovu načíst.

5.7.2 Editace porušených souborů

Validita ELF binárních souborů je kontrolována na základě offsetů obsažených v hlavičkách. Takové chyby se nemusí projevit hned v místě poruchy, ale až o pár bloků později, stejně jako je vidět na obrázku 5.1. Chyba následuje hned za ELF hlavičkou, kdy je za ni přidán jeden byte obsahující nuly, takže zbytek souboru je posunutý o jeden byte. V ideálním případě by se měla chyba projevit hned u prvního načteného segmentu, ale posunutí dat o jeden byte způsobilo, že první dva segmenty obsahují offsety, které neodkazují mimo hranice souboru. Až ve třetím segmentu je načtený offset větší než velikost souboru a tím je dokázáno, že je soubor poškozený.

Pro ukázku editace porušeného souboru je vybrán soubor, který je zobrazen i na obrázku 5.1. V tomto případě je oblast poškození známá. Všechna data od bytu 65 až do konce jsou posunuta o jeden byte. Pro lepší představu bude ukázka podpořena zobrazením grafu sekcí mezi jednotlivými kroky editace ELF binárního souboru na obrázku 5.4.

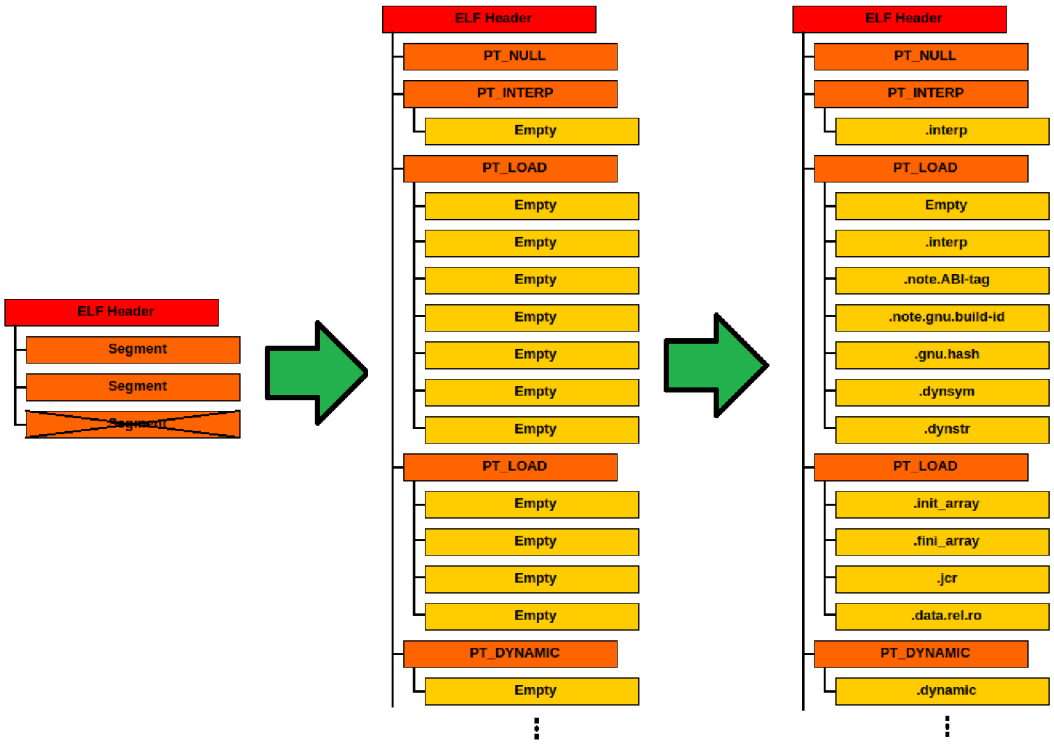
První krok úpravy ELF binárního souboru je posunout v ELF hlavičce offsety odkazující na programovou hlavičku a na hlavičku sekcí. Na obrázku 5.4 můžete sledovat, že z neúplného grafu sekcí vlevo se stal úplný graf, ale nejsou na něj aplikovány jména sekcí, které jsou uložena v paměti souboru. To znamená, že programovou tabulku a tabulku sekcí aplikace načetla správně, ale offsety jednotlivých sekcí jsou stále posunuté o jeden byte.

Jako druhý krok je pro lepší ukázku provedena úprava offsetu tabulky řetězců obsahující jména sekcí. Z ELF hlavičky je možné zjistit, že tabulka řetězců se skrývá pod indexem 37. Proto po úpravě offsetu v tabulce sekcí, který odpovídá tabulce řetězců se aplikují jména na jednotlivé sekce v grafu sekcí.

Takto by to bylo možné provést i u ostatních sekcí, ale jak bylo zmíněno v sekci 5.7.1, existují sekce, které obsahují informace o realokaci sekcí, takže pro validní úpravu by bylo nutné upravit celý obsah sekce, která může obsahovat stovky takových informací. Toto všechno je možné pouze za předpokladu, že uživatel zná oblast poškození ELF binárního souboru, což ve spoustě případů poškozených ELF souborů není pravda.

5.8 Testování

Aplikace byla vyvíjena v prostředí Qt Creator na virtualizovaném Linuxu, distribuce Ubuntu (verze 14.04.03 LTS). Pro vývoj byl použit programovací jazyk C++, rozšířený frameworkem Qt pro tvorbu GUI. Další operační systém, na kterém byla aplikace vyzkoušena je Microsoft Windows 8.1.



Obrázek 5.4: Jednotlivé kroky úpravy z pohledu grafu sekcí

Pro testování výsledného produktu bylo použito manuální testování. Ručně byly narušeny ELF binární soubory. Ty pak byly otevírány pomocí výsledné aplikace a následně upravovány a ukládány. Díky testování byly odhaleny chyby ve zdrojovém kódu, jako různé chyby v alokaci paměti, překlepy, přistupování do nealokované paměti a jiné.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo seznámení se s formátem ELF. Navrhnout a implementovat aplikaci, která dokáže zobrazit obsah ELF binárních souborů. Aplikace se musí umět vyrovnat s poškozenými ELF binárními soubory a musí podporovat ruční editaci. Aplikace by měla být implementována multiplatformně. Poslední bod zadání byl pak navrhnout vylepšení výsledné aplikace do budoucna.

Seznámení se s formátem ELF nebylo obtížné. Krom jedné malé dezinformaci v dokumentaci k ELF formátu byla tato dokumentace velmi přínosná. Návrh aplikace, po pár konzultacích s vedoucím práce a s technickým konzultantem, byl vytvořen. Hlavním cílem bylo vytvořit přehledný graf sekcí, který zobrazí vnitřní strukturu ELF binárních souborů. Tento graf pak měl být podpořen dalšíma dvěma pohledy. Jeden zobrazující vlastnosti sekcí a druhý obsahy sekcí. Tento cíl se povedlo splnit. Graf sekcí funguje a zbylé dva pohledy také. Zde nastal jen problém s časem, kdy se autor této práce zdržel při zkoumání knihovny Qt a implementaci výsledného grafu.

Editaci ELF binárních souborů, které byly předtím narušeny, nebyl už takový problém implementovat s ohledem na nabyté znalosti knihovny Qt a ELF formátu. Tato část byla implementována celkem v pořádku.

Předposlední bod byl asi nejobtížnější. Nejprve bylo s vedoucím práce a technickým konzultantem domluveno, že další OS, na kterém půjde spustit výsledná aplikace, bude Mac OS X. Ale tento OS autor neměl k dispozici a Mac OS X nelze virtualizovat. Proto bylo připuštěno, že bude aplikace implementována na OS Microsoft Windows. Použité knihovny, pro převádění strojového kódu však na Windows nefungovaly, proto verze pro spuštění na OS Microsoft Windows postrádá zobrazení spustitelných instrukcí.

Poslední bod v zadání byl navrhnout vylepšení. Výsledná aplikace funguje podle zadání, tak jak má. Avšak GUI aplikace je velmi strohé. Uživateli chybí větší variabilita v nastavení GUI. Aplikace nepodporuje klávesové zkratky pro urychlení práce s aplikací. Jak bylo zmíněno výše, aplikace funguje plně jen na OS Linux, proto by bylo vhodné rozšířit funkcionalitu aplikace i na Mac OS X a vyladit aplikaci na OS Microsoft Windows.

Literatura

[1] Albatross: *Dokumentace jazyka C/C++*. [Online; navštíveno 28.12.2016].
URL http://www.cplusplus.com/

[2] Burkholder, J.: *Utilita elfdump*. [Online; navštíveno 02.11.2016].
URL https://docs.oracle.com/cd/E19683-01/817-0689/6mgfkpcn6/index.html

[3] Lamikhov-Center, S.: *Dokumentace knihovny ELFIO*. [Online; navštíveno 18.11.2016].
URL http://elfio.sourceforge.net/elfio.pdf

[4] Linux Foundation: *Knihovna elf.h*. [Online; navštíveno 21.11.2016].
URL
https://github.com/torvalds/linux/blob/master/include/uapi/linux/elf.h

[5] Ludd, N.; Frysinger, M.; Groffen, F.: *Utilita scanelf*. [Online; navštíveno 02.11.2016].
URL https://linux.die.net/man/1/scanelf

[6] Qt-Company: *Dokumentace knihovny Qt pro tvorbu GUI*. 1991, [Online; navštíveno 05.11.2016].
URL http://doc.qt.io/

[7] Qt-Project: *Dokumentace grafického editoru pro tvorbu grafických aplikací s využitím knihovny Qt*. 2009, [Online; navštíveno 05.11.2016].
URL http://doc.qt.io/qtcreator/

[8] Radare: *Knihovna Radare2*. [Online; navštíveno 28.11.2016].
URL https://github.com/radare/radare2

[9] Smart, J.; Zeitlin, V.; Dunn, R.; aj.: *Dokumentace knihovny wxWidget*. [Online; navštíveno 12.01.2017].
URL http://docs.wxwidgets.org/3.0/

[10] The Go Authors: *Dokumentace knihovny Capstone*. [Online; navštíveno 16.05.2017].
URL http://www.capstone-engine.org/documentation.html

[11] The Go Authors: *Přehled architektur a OS objevujících se v ELF hlavičce*. [Online; navštíveno 16.05.2017].
URL https://github.com/google/rekall/blob/master/tools/windows/winpmem/executable/elf.h

[12] Turner, P. D.; Team, T. C. D.: *Dokumentace knihovny CEGUI pro tvorbu GUI*. 2004, [Online; navštíveno 05.11.2016].
URL http://static.cegui.org.uk/docs/0.8.7/

[13] Unix-System-Laboratories: *Utilita readelf*. [Online; navštíveno 02.11.2016].
URL https://linux.die.net/man/1/readelf

[14] Unix-System-Laboratories: *Dokumentace formátu ELF binárních souborů*. 1999, [Online; navštíveno 11.10.2016].
URL http://www.skyfree.org/linux/references/ELF_Format.pdf

[15] Weyergraf, S.; Biallas, S.: *Manuál textového editoru HTEditor pro úpravu ELF binárních souborů*. 1999, [Online; navštíveno 01.11.2016].
URL http://hte.sourceforge.net/readme.html

[16] Xinuos Inc.: *Přehled architektur a OS objevujících se v ELF hlavičce*. [Online; navštíveno 15.05.2017].
URL http://www.sco.com/developers/gabi/latest/ch4.eheader.html

Přílohy

Příloha A

Obsah přiloženého paměťového média

Po načtení paměťového média naleznete tuto adresářovou strukturu:

- Linux/ – Složka obsahující soubory pro OS Linux
- Linux/src/ – Složka obsahující zdrojové soubory pro vytvoření spustitelného programu
- Linux/ELFviewer – Spustitelný soubor
- Win/ – Složka obsahující soubory pro OS Windows
- Win/src/ – Složka obsahující zdrojové soubory pro vytvoření spustitelného programu
- text/ – Složka obsahující zdrojové soubory pro vytvoření dokumentace
- ELFviewerTest – Porušený spustitelný soubor z obr 5.1, který je použitý pro demonstraci editace porušených souborů

Příloha B

Manuál

Pros spuštění výsledné aplikace na Linuxu je nutné si nainstalovat aplikaci Qt Creator.

Na OS Linux otevřete v Qt Creatoru soubor `ELFviewer.pro`, který najdete ve složce `Linux/src`. Otevře se vám projekt v Qt Creatoru. V levém dolním rohu zmáčkněte tlačítko Run/Spustit. Spustí se vám výsledná aplikace a ve vedlejší složce build-ELFviewer... naleznete soubor `Makefile`, pomocí kterého můžete aplikaci zkompilovat a spustitelný soubor `ELFviewer`, který vytvořil Qt Creator.

Pro spuštění výsledné aplikace na Windows je nutné si nainstalovat aplikaci Qt Creator.

Na OS Windows otevřete v Qt Creatoru soubor `ELFviewer.pro`, který najdete ve složce `Linux/src`. Otevře se vám projekt v Qt Creatoru. V levém dolním rohu zmáčkněte tlačítko Run/Spustit. Spustí se vám výsledná aplikace.