VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

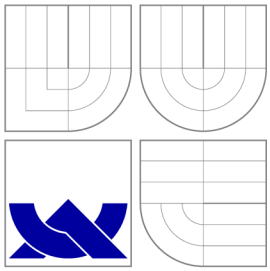
NÁSTROJE PRO KONVERZI FORMÁTŮ SPUSTITELNÝCH SOUBORŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

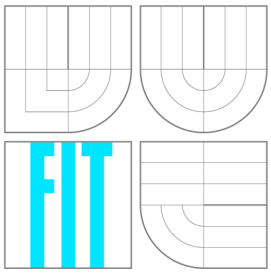
AUTOR PRÁCE
AUTHOR

PETER MATULA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJE PRO KONVERZI FORMÁTŮ SPUSTITELNÝCH SOUBORŮ

TOOLS FOR EXECUTABLE FILE FORMAT CONVERSIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETER MATULA

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Ing. HRUŠKA TOMÁŠ, CSc.

BRNO 2011

Abstrakt

Táto práce popisuje metody a postupy používané ku konverzi formátů objektových souborů. Představuje několik obecně používaných formátů (ELF, PE, E32Image, DEX) a objektový formát projektu Lissom (LOFF). Obsahuje základní informace o knihovnách manipulujících tyto formáty a popis nové knihovny spravující E32Image. Primárním ukolem je implementace programu pro konverzi souborů mezi obecně používanými formáty a formátem LOFF. Tento problém je řešen mapováním všech kritických informací z jedné struktury formátu do druhé. Za tímto účelem bylo nutné upravit a rozšířit některé vlastnosti formátu projektu Lissom. Výsledek je program využívající pluginovací systém, schopný vytvářet validní a spustitelné soubory ve zmíněných formátech.

Abstract

This paper describes methods and procedures used for object file format conversions. It introduces several commonly used formats (ELF, PE, E32Image, DEX) and project Lissom's object file format (LOFF). It contains basic information about libraries manipulating these formats and a description of a new library managing E32Image. The primary objective is to implement a program converting files between common formats and LOFF. This problem is solved by mapping all critical information from one format structures to another. To accomplish this task, it was necessary to modify and extend some features of Lissom object format. The result is the plugin based application capable of creating valid and runnable executable files in mentioned formats.

Klíčová slova

Konverze objektových souborů, objektové soubory, zpětný překlad, dekompilátor, disasembler, Lissom, LOFF, ELF, PE, E32Image, DEX, BFD.

Keywords

Object file conversion, object file formats, decompilation, decompiler, disassembler, Lissom, LOFF, ELF, PE, E32Image, DEX, BFD.

Citace

Peter Matula: Nástroje pro konverzi formátů spustitelných souborů, bakalářská práce, Brno, FIT VUT v Brně, 2011

Nástroje pro konverzi formátů spustitelných souborů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Prof. Tomáša Hrušky.
Další informace mi poskytli Ing. Jakub Křoustek, Ing. Adam Husár a Ing. Zdeněk Přikryl. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Peter Matula
18. května 2011

Poděkování

Na tomto místě bych rád poděkoval mému vedoucímu Prof. Tomášu Hruškovi a mému konzultantovi Ing. Jakubu Křoustkovi za odborné vedení, poskytnuté rady a možnosť zapojiť se do projektu Lissom. Dále bych chtěl poděkovat členům projektu za mnoho užitečných podnětů.

Obsah

Kapitola 1

Úvod

V súčasnej dobe sú *vstavané systémy* (anglicky embedded systems) súčasťou života väčšiny ľudí a môžeme ich nájsť takmer vo všetkých, čo len trochu komplikovanejších elektronických zariadeniach. Výroba takýchto zariadení prirodzene prináša obrovské zisky, konkurencia je veľká a tlak na rýchly a lacný vývoj nových systémov rastie. To ako rýchlo dokáže výrobca vyvinúť optimálne riešenie vo veľkej miere rozhoduje o úspechu alebo neúspechu. Práve preto sa hľadajú inovatívne spôsoby, ako pristupovať k výrobe nových architektúr. Jedným z riešení je takzvaný *hardware-software codesign*, spoločný proces návrhu hardwaru a softwaru.

Proces návrhu začína modelovaním *procesoru* (anglicky Central Processing Unit, CPU) v niektorom z *jazykov pre popis architektúr* (anglicky Architecture Description Languages, ADL). Najčastejšie sa používajú zmiešané jazyky umožnujúce namodelovať zdroje, model správania, model časovania a zároveň inštrukčnú sadu. Na základe takéhoto modelu sa vygenerujú špecifické nástroje. Generovanie prebieha automaticky, teda veľmi rýchlo v porovnaní s manuálnym vývojom. Cieľom sú minimálne *asemblér* a *linker*, potrebné pre tvorbu programov spustiteľných na modelovanom procesore. Na rovnakom princípe však môžu byť vygenerované aj iné nástroje ako napríklad *ladiaci nástroj* alebo *prekladač*. Pre testovanie kvality navrhnutého procesoru je potrebné sledovať jeho výkon pri spracovaní programu vytvoreného pomocou generovaných nástrojov. Aby nebolo nutné v každej iterácií procesor fyzicky vyrobiť, používa sa *simulátor*. Ten dokáže napodobniť činnosť CPU pri vykonávaní programu. Práca procesoru sa vyhodnotí a v prípade neuspokojivého výsledku pokračuje vývoj ďalšou iteráciou – vytvorením nového modelu.

Výhody takéhoto prístupu sú zjavné. Automatizované generovanie nástrojov urýchli a ulahčí prácu a simulácia odstráni nutnosť nákladnej výroby testovacích procesorov. Práve vývojom podobného nástroja sa zaoberá aj projekt *Lissom*, pre potreby ktorého vznikla táto práca.

Cieľom bakalárskej práce je vytvoriť program schopný obojsmernej konverzie medzi obecne používanými objektovými formátmi a formátom používaným nástrojmi projektu Lissom. Pričom transformácia by sa mala zamerať primárne na spustiteľné súbory.

Informácie o projekte Lissom a úlohe, ktorú v ňom práca zohráva sú uvedené v kapitole 2. V krátkosti je vysvetlený pojem *dekompilátor*, ako hlavný užívateľ vzniknutej aplikácie.

Nasleduje popis objektového formátu projektu Lissom a predstavenie obecných formátov operačných systémov *UNIX*, *Windows*, *Symbian* a *Android*.

Štvrtá kapitola popisuje zmeny navrhnuté a implementované autorom práce na formáte projektu Lissom za účelom zdokonalenia jeho schopnosti zaznamenať iné formáty.

Prípady úspešného využitia konverzie objektových súborov v minulosti a knižnice

použité výsledným programom za týmto účelom sú v krátkosti predstavené v kapitole 5. Najväčšia pozornosť sa venuje popisu knižnice *E32lib*, vytvorenej v rámci bakalárskej práce za účelom uľahčenia manipulácie *E32Image* súborov.

Ďalšie dve kapitoly sa venujú implementácií programu, pluginov a testovaniu jeho schopností. Popis sa podrobne zameriava na navrhnuté rozhranie pre vývoj pluginov, jednotlivé konverzie, ich špecifiká a problémy ktoré nastali.

V závere sú diskutované dosiahnuté výsledky, praktické využitie a budúci vývoj aplikácie.

Kapitola 2

Projekt Lissom

Projekt *Lissom* sa zaoberá vytvorením univerzálnych nástrojov pre návrh nových procesorov. Ako hlavný programovací jazyk bol zvolený C/C++ poskytujúci efektívnu multiplatformovosť. Pre popis architektúry navrhovaného CPU je používaný jazyk *ISAC* [9] (vychádza z jazyka *LISA*) vyvinutý pre potreby projektu. Z modelu v tomto jazyku vygeneruje prekladač *XML* dokument popisujúci navrhnutý procesor. Z dokumentu sa ďalej automaticky generuje asemblér, disasemblér, prekladač, ladiaci nástroj, simulátor, atď. V budúcnosti pribudne dekompilátor, ktorý je v rannej fáze vývoja. Schopnosť programu vygenerovať všetky uvedené nástroje len na základe vstupu vo forme popisu architektúry výrazne uľahčí a urýchli vývoj nových procesorov.

2.1 Cieľ práce

Ako bolo uvedené vyššie, výstupom programu vyvíjaného v projekte Lissom je niekoľko nástrojov, ktoré nejakým spôsobom pracujú s objektovými súbormi. Pre potreby projektu bol vyvinutý nový objektový formát *LOFF*[1], popísaný v podkapitole 3.1. Výhodou vytvorenia vlastného, interného, formátu je možnosť navrhnúť a prispôsobiť ho tak, aby bol vhodný na riešenie konkrétnych úloh v rámci projektu. Problémom je, že takto vytvorený formát sa nevyužíva nikde v priemysle, a preto je nutný jeho preklad do obecne používaných formátov a naspäť. Práve touto úlohou sa zaoberá táto práca, na základe ktorej vznikol program nazvaný *bintran* (*bin*ary *tran*slator, prekladač binárnych súborov) vykonávajúci konverziu spustiteľných súborov. Program nájde hlavné využitie pri predspracovaní vstupov pre vznikajúci dekompilátor a pri testovaní binárnej kompatibility.

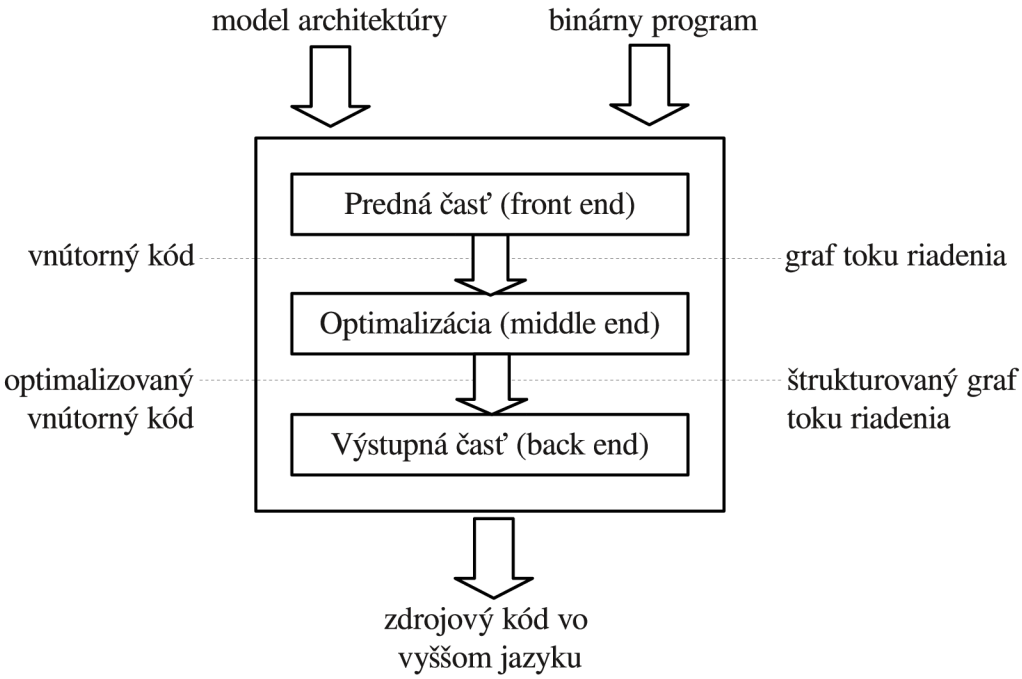
2.2 Dekompilátor

Podľa [7]:

> *„Dekompilácia je programová transformácia, ktorá má za cieľ zo vstupného programu vytvoriť kód vo vyššom programovaciom jazyku“.*

Program vykonávajúci túto činnosť sa nazýva dekompilátor a jeho štruktúra je zobrazené na obrázku 2.1.

[1] *Lissom object file format* – dočasné pomenovanie pre potreby tejto práce. V čase písania práce nebol formát oficiálne pomenovaný.



Obrázok 2.1: Štruktúra obecného dekompilátoru. Obrázok je prevzatý z [7] a upravený.

Jedným z cieľov projektu Lissom je vytvoriť práve obecný dekompilátor, schopný dekompilovať ľubovoľný vstupný program za pomoci modelu procesoru, pre ktorý bol preložený. Súbory pre rôzne architektúry však používajú rôzne objektové formáty a nebolo by praktické, keby front-end dekompilátoru prijímal a spracovával každý z nich.

Preto dekompilátor očakáva na vstupe jedine súbor vo formáte LOFF, ktorý mohol vzniknúť z obecne používaného formátu práve za pomoci nástroja pre konverziu spustiteľných súborov.

2.3 Testovanie binárnej kompatibility

V rámci projektu Lissom vznikol model procesoru použitého v platforme *FITkit*. Pomocou modelu boli vygenerované nástroje (prekladač, simulátor, atď.) a tie následne vytvorili programy pre platformu vo formáte LOFF. Súbory boli aplikáciou bintran preložené do formátu ELF, nahrané do FITkitu a spustené. Konverzia prebehla aj opačným smerom. Programy bežne používané na platforme, boli preložené do Lissom formátu a následne vykonané v simulátore.

Aplikácia bintran teda bola použitá za účelom overenia funkčnosti ostatných nástrojov projektu.

Kapitola 3

Formáty objektových súborov

[8], z ktorej vychádza táto kapitola, definuje:

> *„Formát objektového súboru popisuje štruktúru súborov používaných pre uloženie objektového kódu a odvedených informácií. Typicky sa jedná o výstup prekladaču alebo asembléru.“*

Existuje veľké množstvo takýchto formátov, líšiacich sa vnútornou syntaxou a komplexnosťou zaznamenaných informácií. Všeobecne však môžeme povedať že objektový súbor obsahuje tieto druhy dát:

- *Hlavička* – uchováva základné informácie o súbore ako napríklad identifikáciu formátu, dátum vytvorenia, veľkosť, organizáciu ostatných častí súboru a iné.
- *Objektový kód* – je binárna reprezentácia inštrukcií.
- *Relokačné záznamy* – slúžia pre správne umiestnenie nevyriešených odkazov alebo pri načítaní programu do pamäte na inú adresu ako je tá, pre ktorú bol zostavený. Podľa [11] je relokovanie proces spájania symbolických referencií so symbolickými definíciami. Tento proces je vykonávaný linkerom, ktorý načíta všetky vstupné objektové súbory, nájde v nich relokačné záznamy a symbolické informácie a pospája ich do spustiteľného súboru.
- *Symboly* – definované v tomto module a symboly importované z iných modulov.
- *Ladiace informácie* – nie sú nutné pri linkovaní alebo spustení súboru, sú využívané ladiacim nástrojom.

[8] delí objektové súbory podľa účelu na:

- *Spustiteľné* – je možné ich načítať do pamäte a spustiť ako program, typicky neobsahujú relokácie a symboly (ak nie je nutné linkovanie za behu programu).
- *Linkovateľné* – vznikajú ako výsledok prekladu modulov, sú použité ako vstup linkeru. Ten z nich vytvorí spustiteľné súbory alebo knižnice. Z tohto dôvodu obsahujú veľké množstvo relokácií a symbolov.
- *Knižnice* – je možné ich načítať do pamäte spolu s programom.

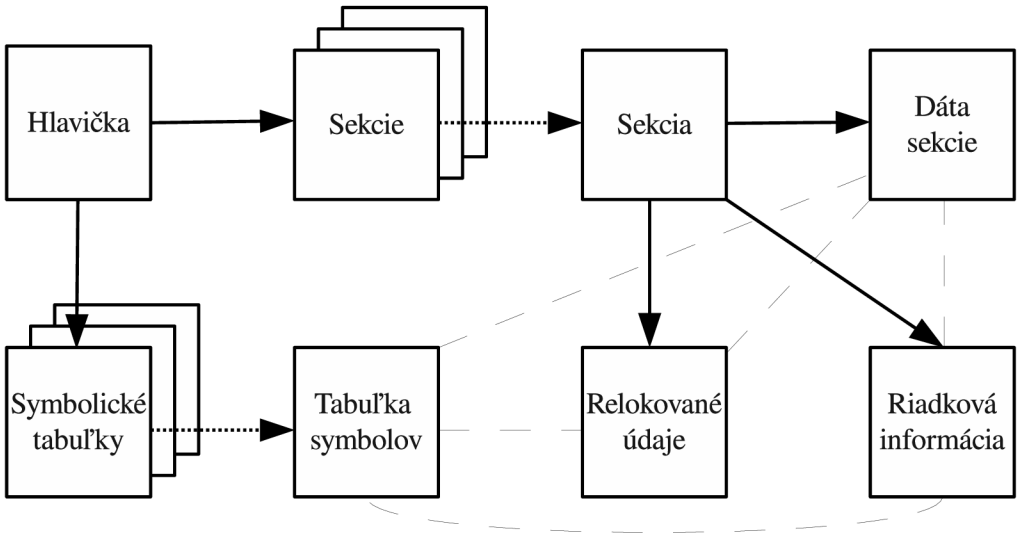
3.1 LOFF

Lissom Object File Format (LOFF) je textový, objektový formát navrhnutý pre potreby projektu Lissom. Cieľom návrhu bolo vytvoriť jednoduchý, priamo čitateľný formát vhodný na reprezentáciu celej rady iných objektových formátov. V tejto podkapitole, spracovanej zo [6], je popísaná verzia LOFFu v čase začiatku práce autora na projekte. Úpravy formátu prameniace z tejto práce sú popísané v podkapitole 4. Základné vlastnosti sú nasledujúce:

- Nezávislosť na šírke slova inštrukcie.
- Nezávislosť na spôsobe uloženia čísiel (*little-endian*, *big-endian*).
- Nezávislosť na charaktere inštrukčnej sady.
- Nezávislosť na spôsobe uloženia, druhu pamäti a prístupu do nej.

3.1.1 Štruktúra LOFF formátu

Obrázok 3.1 zobrazuje väzby medzi časťami LOFF formátu. Podrobnejšia schéma je v prílohe A.



Obrázok 3.1: Väzby v LOFF formáte. Obrázok prevzatý z [6] a upravený.

Hlavička

Tak ako celý formát je aj hlavička veľmi minimalistická. Začína magickým reťazcom identifikujúcim LOFF, nasleduje hodnota udávajúca počet bitov v slove a spôsob uloženia slov v pamäti. Sú to jediné dve hodnoty popisujúce cieľovú architektúru, nenájdeme tu žiadnu identifikáciu konkrétnej rodiny alebo typu procesoru. Ďalšími informáciami sú časové razítko, príznaky súboru, informácie o počte sekcií, počte tabuliek symbolov a adrese tabuľky symbolov. Príznaky nesú informáciu o obsahu a účele súboru.

Sekcie

Priamo za hlavičkou nasleduje zoznam hlavičiek logických častí, sekcií. Každá obsahuje meno, adresu v pamäti, na ktorú sa sekcia pri vykonávaní programu uloží, príznaky a informácie o dátach, relokovaných údajoch a riadkových informáciach asociovaných s danou sekciou.

Typ sekcie je určený jedným z nasledujúcich príznakov:

- ***T**ext* – spustiteľný kód.
- ***D**ata* – inicializované dáta.
- ***B**ss* – neinicializované dáta.

Samotné dáta nasledujú až po zozname všetkých hlavičiek. V súbore sú uložené ako súvislý blok binárnych hodnôt. Adresa začiatku každého datového bloku je uložená v príslušnej hlavičke sekcie.

Relokácie

Relokácie v LOFF formáte sú vytvárané nástrojmi projektu Lissom pri preklade zdrojových súborov a sú spracované Lissom linkerom pri zostavovaní výsledného spustiteľného súboru. Z tohto dôvodu sa jedná o unikátne záznamy, ktorých formát odráža potreby projektu a jeho nástrojov. Výsledný spustiteľný súbor relokácie väčšinou neobsahuje a nástroj vytvorený na základe tejto práce relokačné záznamy medzi formátmi typicky neprekladá (prekladané sú len modulom *bfdtoloff*, viz 6.4)

Každá relokácia obsahuje adresu v dátovej sekcií, ktorá bude upravená, typ relokácie, veľkosť relokácie, offset, počet bitov pre pravý posun a masku. Typ relokácie určuje, či sa jedná o absolútnu alebo relatívnu relokáciu a tiež spôsob, ako použiť ostatné hodnoty.

Čísla riadkov

Tieto informácie slúžia k prepojeniu riadkov v zdrojovom kóde s adresou postupnosti inštrukcií v preloženom objektovom súbore. Údaje slúžia len pre ladiace účely a nemajú vplyv na vykonanie programu.

Celý blok dát začína údajom o počte záznamov, nasledovaný konkrétnymi záznamami obsahujúcimi číslo riadku v pôvodnom zdrojovom súbore, názov zdrojového súboru a relatívnu adresu v dátach asociovanej sekcie.

Tabuľka symbolov

Tabuľky symbolov obsahujú symboly potrebné pre podporu návestí a ich počet nie je formátom obmedzený. Aktuálne nástroje ale predpokladajú uloženie všetkých symbolov len v jednej tabuľke a ani ostatné formáty väčšinou nepoužívajú viac tabuliek.

Každý symbol obsahuje názov, typ, index asociovanej sekcie, adresu symbolu v sekcii a prípadnú rozširujúcu informáciu. Typ symbolu je jeden z nasledujúcich:

- *p**U**blic* – verejný symbol definovaný v tomto súbore, je verejne viditeľný – je možné sa naň odkazovať z ostatných modulov.
- *p**R**ivate* – privátny symbol definovaný v tomto súbore, na ktorý sa nedá odkázať.
- *e**X**tern* – externý symbol nedefinovaný v tomto module, je očakávaný v inom súbore.

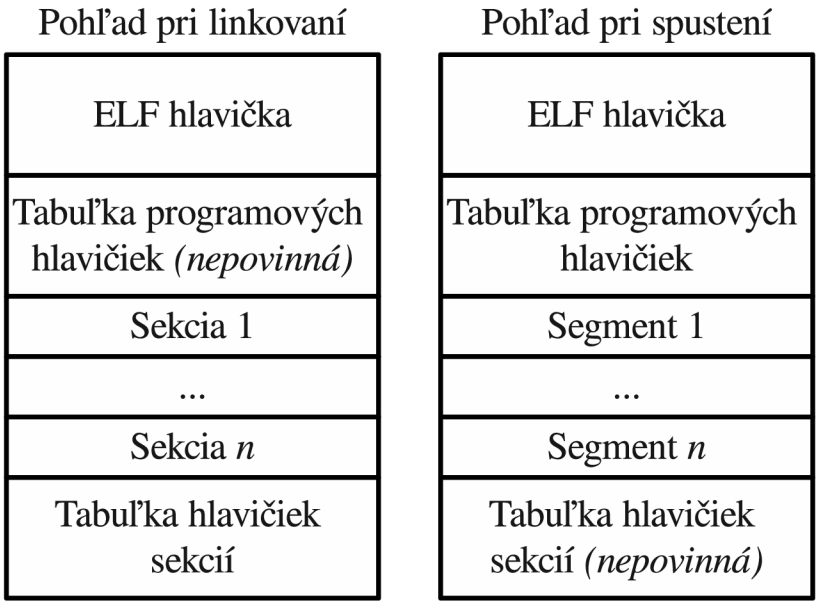
3.2 UNIX ELF

Executable and Linkable Format (ELF), je v súčasnosti štandardom používaným vo všetkých *UNIX-like* operačných systémoch. Prvýkrát bol nasadený v *UNIX System V* ako náhrada tradičného *a.out* formátu. Ten nepodporoval dynamické linkovanie, krížový preklad a ďalšie moderné funkcie. Jedná sa o flexibilný formát, ktorý nie je spätý so žiadnou konkrétnou architektúrou alebo procesorom, vďaka čomu sa rýchlo rozšíril na množstvo rôznych platform. Existujú tri základné typy ELF súborov odpovedajúce trom typom objektových súborov podľa účelu: spustiteľné, linkovateľné a zdieľané (knižnice).

Podkapitola je spracovaná na základe [8][11].

3.2.1 Štruktúra ELF formátu

ELF súbory sú z pohľadu prekladača, asembléru a linkeru videné ako skupina sekcií (pohľad pri linkovaní), definovaných v tabuľke hlavičiek sekcii. Na druhej strane loader (program načítava aplikácie do pamäte a pripravuje ich na exekúciu) pracuje so súborom ako so skupinou segmentov (pohľad pri spustení), definovaných v tabuľke programových hlavičiek. Sekcie sú určené pre ďalšie spracovanie, segmenty pre načítanie do pamäte a následné vykonanie. Oba pohľady sú zobrazené na obrázku 3.2.



Obrázok 3.2: Schéma ELF súboru.

Hlavička

Na začiatku ELF súboru sa nachádza hlavička. Prvá hodnota identifikuje ELF formát, druhá určuje o aký typ súboru sa jedná (spustiteľné, linkovateľné, zdieľané, CPU špecifické, bez typu). Ďalšie hodnoty popisujú cieľovú architektúru, verziu formátu, vstupný bod pre vykonanie programu, príznaky súboru, veľkosti a pozície ostatných logických častí.

Sekcie

Tabuľka hlavičiek sekcií je pole štruktúr popisujúcich sekcie súboru. Každý záznam obsahuje veľké množstvo informácií. Najdôležitejšie sú meno, typ, príznaky, offset v súbore, velkosť a adresa, kde je treba sekciu uložiť pri načítaní do pamäte. Sekcie obsahujú všetky informácie objektového súboru, okrem tých v hlavičke, tabuľke hlavičiek sekcií alebo v tabuľke programových hlavičiek. Pole `sh_flags` špecifikuje príznaky sekcie a može obsahovať hodnoty:

- `SHF_WRITE` – obsahuje dáta, ktoré môžu byť počas exekúcie prepísané.
- `SHF_ALLOC` – dáta budú načítané pri exekúcií programu.
- `SHF_EXECINSTR` – obsahuje vykonateľné inštrukcie.
- `SHF_MASKPROC` – všetky bity sú rezervované pre CPU špecifickú sémantiku.

Pole `sh_type` špecifikuje sémantiku sekcie. Z pohľadu predkladu do formátu LOFF sú dôležité tieto hodnoty:

- `PROGBITS` – obsahuje kód programu, dáta a ladiace informácie.
- `NOBITS` – podobné `PROGBITS`, ale sekcia nemá alokované miesto v programe. Používané pre `.bss` sekciu.
- `SYMTAB` a `DYNSYM` – sekcia obsahuje tabuľku symbolov.
- `REL` a `RELA` – obsahuje relokačné informácie.
- `DYNAMIC` a `HASH` – informácie doležité pre linkovanie programu za behu.

ELF neurčuje, koľko sekcií má súbor obsahovať ani to, aké názvy a vlastnosti majú mať. Aj napriek tomu existuje niekoľko špeciálnych sekcií, ktorých sémantika je preddefinovaná a sú očakávané operačným systémom. Najvýznamnejšie z nich sú zobrazené v tabuľke 3.1.

Názov	**Typ**	**Príznaky**	**Popis**
`.text` `.init, .fini`	`PROGBITS`	`SHF_ALLOC` `SHF_EXECINSTR`	Vykonateľné inštrukcie.
`.data`	`PROGBITS`	`SHF_ALLOC` `SHF_WRITE`	Dáta.
`.rodata`	`PROGBITS`	`SHF_ALLOC`	Read-only data.
`.symtab`	`SYMTAB`		Statická tabuľka symbolov.
`.dynsym`	`DYNSYM`	`SHF_ALLOC`	Dynamická tabuľka symbolov.
`.strtab`	`STRTAB`		Tabuľka reťazcov.

Tabuľka 3.1: Špeciálne sekcie formátu ELF.

Tabuľky reťazcov

Obsahujú všetky textové reťazce v súbore. Ak má napríklad symbol názov, jeho textová reprezentácia nie je uložená v tabuľke symbolov spolu s ostatnými informáciami. Tam sa nachádza len hodnota identifikujúca reťazec v odpovedajúcej tabuľke reťazcov.

Tabuľka symbolov

Tabuľky symbolov obsahujú informácie využívané linkerom počas zostavovania spustiteľného programu (staticky alebo dynamicky). Jedná sa o vektor záznamov, z ktorých každý obsahuje meno, adresu symbolu, veľkosť, index asociovanej sekcie a niekoľko ďalších informácií.

Relokácie

Tak ako ostatné dáta v ELF formáte aj relokácie sú uložené v sekciách. Ide o vektory záznamov určujúcich adresy v kóde, ktoré treba opraviť a spôsob, ako majú byť opravené. Existujú dva typy relokačných sekcií:

- `SHT_RELA` – označuje relokácie obsahujúce v zázname takzvaný *addend*, hodnotu použitú pri procese relokovania. Spôsob použitia je zavislý na type relokácie.
- `SHT_REL` – relokácie majú addend uložený v dátach sekcie na relokovanej adrese.

3.3 Windows PE

Portable Executable (PE), je objektový formát pre spustiteľné, linkovateľné a zdieľané súbory (DLL knižnice) používaný v tridsaťdva a šesťdesiatštyri bitových verziách operačných systémov *Windows* už od verzie *Windows NT 3.1*. Jedná sa o upravenú verziu formátu *COFF* používaného v OS UNIX. PE je prenosné na mnoho architektúr ako napríklad *IA-32*, *IA-64*, *x86-64*, *MIPS*, *Alpha*, *PowerPC*, *ARM* a ďalšie.

Podkapitola je spracovaná na základe [8][10].

3.3.1 Štruktúra PE formátu

Na obrázku 3.3 vidíme základnú schému PE súboru. Tak ako ostatné formáty aj PE začína hlavičkou a pozostáva z logických častí nazývaných sekcie. Nie všetky sekcie sú povinné a ich prítomnosť je závislá na type objektového súboru.

MS-DOS Stub

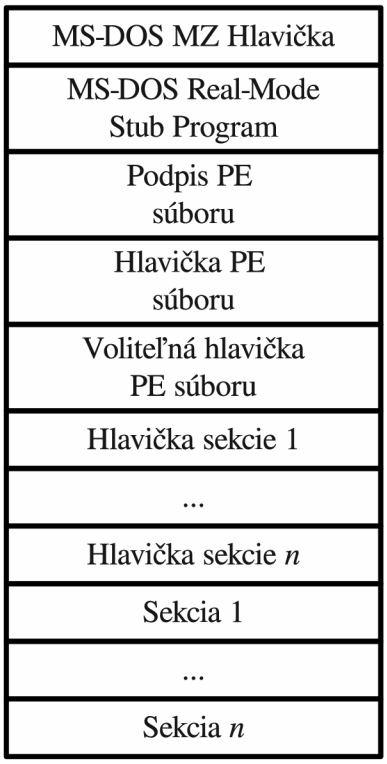
Jedná sa o aplikáciu bežiacu pod OS *MS-DOS* umiestnenú na začiatku každého spustiteľného súboru. Účelom je vytlačiť správu: *„This program cannot be run in DOS mode“* pri pokuse o exekúciu v OS MS-DOS.

Podpis PE súboru

Za MS-DOS Stub sa v spustiteľných súboroch nachádza štvor-bajtová identifikácia PE formátu: *„PE \0\0“*.

PE hlavička

Hlavička sa nachádza na začiatku linkovateľného súboru alebo za podpisom v spustiteľnom súbore. Obsahuje identifikáciu cieľovej architektúry, počet sekcií, dátum vytvorenia, príznaky súboru, veľkosť voliteľnej hlavičky, adresu tabuľky symbolov a počet symbolov. Typ súboru (spustiteľný, knižnica, linkovateľný a ďalšie) je príznakom určený práve na tomto mieste.



Obrázok 3.3: Schéma PE súboru.

Voliteľná hlavička

Aj napriek názvu sa musí nachádzať vo všetkých spustiteľných súboroch. Jej účel je poskytovať informácie potrebné pre načítanie súboru do pamäte. Hlavička obsahuje svoj vlastný podpis určujúci, či sa jedná o formát PE32 (tridsaťdva bitov) alebo PE32+ (šesťdesiatštyri bitov). Ďalej môžeme túto časť rozdeliť na:

- Štandardné informácie potrebné pre načítanie a exekúciu programu ako napríklad adresa vstupného bodu alebo veľkosť inicializovaných a neinicializovaných dát.
- Rozširujúce informácie potrebné pre načítanie v prostredí OS Windows.
- Informácie o umiestnení dát, napríklad tabuliek importovaných a exportovaných symbolov.

Hlavičky sekcií

Nasleduje tabuľka hlavičiek pre každú sekciu v súbore. Záznam obsahuje informácie popisujúce danú sekciu: názov, príznaky, veľkosť, adresa dát, informácie o relokáciach a číslach riadkov.

Tak ako pri popise formátu ELF je aj pri PE nutné pochopenie základných príznakov sekcií za účelom správnej transformácie *do* a *z* formátu LOFF. Príznaky odpovedajúce typom sekcií formátu LOFF sú nasledujúce:

- `IMAGE_SCN_CNT_CODE` – sekcia obsahuje spustiteľný kód.

- `IMAGE_SCN_SNT_INITIALIZED_DATA` – sekcia obsahuje inicializované dáta.
- `IMAGE_SCN_SNT_UNINITIALIZED_DATA` – sekcia obsahuje neicializované dáta.
- `IMAGE_SCN_MEM_EXECUTE` – sekcia môže byť vykonaná ako kód programu.
- `IMAGE_SCN_MEM_READ` – sekcia môže byť načítaná.
- `IMAGE_SCN_MEM_WRITE` – sekcia môže byť prepísaná.

Špecialne sekcie

Aj v PE sa nachádzajú sekcie so všeobecne známym významom, očakávané nástrojmi pracujúcimi s objektovými súbormi:

- *Exportované symboly* – tabuľka symbolov (nachádzajúca sa v sekcií `.edata`) definovaných v tomto module a viditeľných (použiteľných) inými modulmi. Na symbol je možné sa odkazovať jeho menom alebo ordinálnym číslom.
- *Importované symboly* – tabuľka symbolov (nachádzajúca sa v sekcií `.idata`) definovaných v inom module (DLL knižnici). Všetky odkazy musia byť vyriešené pred skutočným vykonaním kódu. Symboly sú identifikované menom knižnice v ktorej sa nachádzajú a svojim ordinálnym číslom alebo menom.
- *Tabuľka zdrojov*
- *Thready a lokálne sklady*
- *Fixups* (opravy) – v prípade, že je aplikácia načítaná do pamäte na iné adresové miesto, pre aké bola zlinkovaná je nutné určiť všetky adresy v súbore, ktoré musia byť opravené. Táto sekcia nesie zoznam všetkých takýchto miest v súbore. Ak objektový súbor neobsahuje fixup sekciu, musí byť vždy načítaný na adresu predurčenú pri linkovaní.

3.4 Symbian E32Image

E32Image [2], je formát objektových súborov používaný operačným systémom *Symbian* [5]. Symbian je OS určený pre nasadenie v takzvaných *chytrých telefónoch* (anglicky smartphones). Bol vytvorený firmou *Symbian Ltd.* a v súčasnosti spravovaný spoločnousťou *Nokia*. OS beží výhradne na procesoroch *ARM* a formát E32Image bol vyvinutý z dôvodu prílišnej pamäťovej náročnosti bežných formátov ako ELF alebo PE.

3.4.1 História Symbian OS

Tabuľka 3.2 zobrazuje vývoj OS Symbian a vybrané vlasnosti súvisiace s objektovým formátom. Vidíme, že predchodcom boli OS z rady *EPOC*, vytvorené firmou *Psion*. Od verzie 8.1 bolo nasadené jadro druhej generácie, ktoré rozšírilo hlavičku objektového formátu o informácie o kompresií a data súvisiace s bezpečnosťou aplikácie. Ďalším milníkom je verzia 9.1, ako prvá používajúca nové ARM ABI (*Application Binary Interface*).

Do verzie 9.1 vznikali E32Image prekladom zdrojového kódu do formátu Windows PE a následnou transformáciou programom *petran*[1]. Tá do hlavičky pridala špecifické informácie Symbian OS, mená funkcií sú nahradené ordinálnymi číslami a potenciálne väčšie množstvo PE sekcií je spojené do napevno daných E32Image sekcií.

Od verzie 9.1 sú E32Image súbory založené na UNIX ELF formáte. Program petran je nahradený dvojicou programov *elftran*[2] a *elf2e32* vykonávajúcich rovnakú funkciu. Rozdiel medzi nimi je, že elftran prekladá ELF vo formáte ABIv1 a elf2e32 v novšom ABIv2. Mená funkcií sú aj v tomto prípade nahradené ordinálnymi číslami.

P.I.P.S (*P.I.P.S Is POSIX on Symbian OS*), je snaha priblížiť Symbian k štandardom IT priemyslu. Okrem iného štandard definuje nové cieľové typy ELF transformácie. Vzniknuté súbory sa označujú ako *STDEXE* a *STDDLL*. Jedná sa o objektové súbory typu E32Image použiteľné od verzie 9.3. Obsahujú mená symbolov a za cenu väčšej veľkosti programu umožnujú vyhľadávanie symbolov pomocou týchto mien.

Názov OS	**Rok**	**CPU**	**Jadro**	**E32Image odvodený z**
EPOC16	okolo 1990	i8086	EKA1	–
EPOC32	1997	ARM	EKA1	PE
EPOC Release 1-5	do 2000	ARM	EKA1	PE
Symbian 6.0	2001	ARM	EKA1	PE
Symbian 7.0	2003	ARM	EKA1	PE
Symbian 8.0	2004	ARM	EKA1	PE
Symbian 8.1	2005	ARM	EKA2	PE
Symbian 9.0	2004	ARM	EKA2	PE
Symbian 9.1	2005	ARM	EKA2	ELF
Symbian 9.2	2006	ARM	EKA2	ELF
Symbian 9.3	2006	ARM	EKA2	ELF
Symbian 9.4	2007	ARM	EKA2	ELF
Symbian 9.5	2007	ARM	EKA2	ELF

Tabuľka 3.2: Vývoj Symbian OS.

3.4.2 Štruktúra E32Image formátu

Spoločnosť Nokia nezverejnila oficiálnu špecifikáciu svojho formátu, a tak sú nasledujúce informácie založené na voľne šíriteľných zdrojových kódoch programov petran, elftran, elf2e32 a na analýze vybraných E32Image súborov.

Schéma E32Image je zobrazená na obrázku 3.4. Jedná sa o klasický objektový formát obsahujúci hlavičku, objektový kód, dáta, neinicializavané dáta, relokácie, importované a exportovné symboly.

Hlavička

Hlavička je definovaná triedou `E32ImageHeaderV` odvodenou od tried `E32ImageHeaderComp` a `E32ImageHeader`.

[1] http://wiki.forum.nokia.com/index.php/Petran
[2] http://wiki.forum.nokia.com/index.php/Elftran

Obrázok 3.4: Schéma E32 súboru.

Základná trieda `E32ImageHeader` obsahuje väčsinu informácií o súbore ako napríklad UID (tri hodnoty identifikujúce program), checksum, čas vytvorenia, typ procesoru, veľkosti a adresy sekcií v rámci súboru, typ kompresie a ďalšie. Nikde v hlavičke sa neuvádzajú informácie o poradí bajtov, dĺžke slova a počte bajtov v slove. Implicitne sa predpokladájú tridsaťdva bitové slová obsahujúce štyri bajty uložené vo formáte little-endian.

Trieda `E32ImageHeaderComp` pridáva informáciu o dekomprimovanej veľkosti dát nachádzajúcich sa za hlavičkou, alebo nulu v prípade, že súbor nie je komprimovaný. Pri vytváraní E32Image súboru je možné zvoliť typ kompresie. V súčasnosti sa používajú algoritmy *Deflate – Huffman+LZ77* alebo *Bytepair*.

Trieda `E32ImageHeaderV` pridáva informácie o bezpečnosti súboru, výrobcovi, popisovač výnimiek a dodatočné údaje o tabuľke expotovaných symbolov.

Kódová sekcia

Z obrázku 3.4 je zrejmé, že sekcia označená ako kódová združuje niekoľko podsekcií.

Textová podsekcia obsahuje kód programu. Sekcie s rovnakým názvom a sémantikou môžeme nájsť v rade iných formátov, táto je však jediná svojho druhu v celom E32Image. Vznikla pri transformácií z PE/ELF súboru zlúčením všetkých textových sekcií.

Tabuľka exportovaných symbolov v súčasnosti najpoužívanejšej verzie formátu obsahuje zoznam offsetov exportovaných funkcií. V prípade, že iný modul importuje niektoré z nich, odkazuje sa na ne cez meno súboru, v ktorom sú definované a ordinálne číslo funkcie v tomto zozname. Ako však bolo vyššie uvedené, novšie verzie môžu definovať pomenované symboly.

Tabuľka importovaných symbolov bola súčasťou kódovej sekcie v súboroch založených na Windows PE. V nových verziách založených na UNIX ELF je umiestnená v samostatnej sekcií nachádzajúcej sa za dátami programu.

BSS sekcia

Sekcia obsahuje neinicializované dáta. Jej veľkosť je určená v hlavičke a v súbore sa fyzicky nenachádza.

Dátová sekcia

Dátová sekcia obsahuje inicializované dáta. Tak ako v prípade textovej sekcie je aj ona súhrnom všetkých sekcií obsahujúcich dáta z pôvodného PE/ELF súboru.

Tabuľka importovaných symbolov

Táto sekcia obsahuje pole blokov (počet určený v hlavičke). Každý blok sa skladá z mena importovaného súboru (dynamická knižnica), počtu importovaných symbolov a zoznamu hodnôt s rôznym významom:

- E32Image odvodený z PE – hodnota je ordinálne číslo importovaného symbolu v knižnici.
- E32Image odvodený z ELF – hodnota je offset do textovej sekcie. Na offsete sa nachádza identifikácia importovaného symbolu, ktorá bude nahradená aktuálnou adresou importovanej funkcie.

Tabuľka relokácií

Existujú dve oddelené podsekcie, jedna pre relokácie nachádzajúce sa v kódovej sekcii, druhá pre relokácie v dátovej sekcii. Podsekcia obsahuje údaj o svojej veľkosti, počet relokácií a pole relokačných blokov. Jeden blok urdžuje premenný počet relokačných záznamov nachádzajúcich sa na jednej stránke (4kB). Záznam je šestnásťbitové číslo určujúce adresu v asociavanej sekcii, ktorú treba opraviť v prípade, že je obraz programu načítaný do pamäte na inú počiatočnú adresu, ako je tá pre ktorú bol zlinkovaný. Nejedná sa teda o relokácie používané linkerom na zostavenie spustiteľného súboru, ale o obdobu fixupov známych z formátu PE.

3.5 Android DEX

Dalvik executable (DEX) je formát používaný operačným systémom *Android* určeným pre mobilné zariadania. OS Android je založený na upravenej verzii linuxového jadra, nad ktorou beží *Dalvik virtuálny stroj*. DEX súbor vznikne zo zdrojového kódu napísaného v jazyku *Java*, preloženého štandardným prekladačom do Java bajtkódu. Ten sa následne transformuje nástrojom zvaným *dx*. Výsledkom je súbor vo formáte DEX, obsahujúci Dalvik bajtkód interpretovaný Dalvik virtuálnym strojom. Nejedná se teda o klasický objektový formát obsahujúci inštrukcie pre konkrétny procesor.

Podkapitola je spracovaná na základe dokumentácie k *Android SDK*[3].

[3] http://developer.android.com/sdk/index.html

3.5.1 Štruktúra formátu

Z vyššie uvedeného vyplýva, že štruktúra DEXu sa od ostatných, doteraz predstavených formátov výrazne odlišuje. Nenachádzajú sa tu žiadne sekcie, relokácie alebo symboly. Súbor je v podstate kolekcia tried, ich metód a informácií nutných pre ich definíciu. Zjednodušená schéma formátu je zobrazená v prílohe A.2, zvyšok tejto podkapitoly sa zaoberá popisom klúčových častí.

Hlavička

Hlavička obsahuje identifikáciu formátu, checksum, veľkosť súboru a informácie o pozíciách a veľkostiach jednotlivých podsekcií.

Reťazce

Bezprostredne za hlavičkou sa nachádza zoznam identifikátorov reťazcov, ktorého hodnoty sú offsetmi do datovej sekcie. Offsety ukazujú na položky obsahujúce veľkosť a dáta reťazca.

Dátové typy

Nasleduje zoznam identifikátorov dátových typov. Každý záznam je index do zoznamu reťazcov, z čoho vyplýva že dátové typy sú v konečnom dôsledku len určitá podmnožina reťazcov.

Prototypy

Pre deklaráciu metód je nevyhnutná špecifikácia prototypov. Prototyp má meno, typ návratovej hodnoty a zoznam parametrov. Prvé dve sú uložené ako indexy do príslušných sekcií, zoznam parametrov je identifikovaný offsetom do dátovej sekcie, na ktorom sa začína zoznam dátových typov.

Polia

Položka zoznamu identifikátorov polí obsahuje typ triedy definujúcej toto pole, typ poľa a jeho meno.

Metódy

Identifikátor metódy udržuje jej typ, prototyp a meno. Je nutné poznamenať, že v zozname identifikátorov sa nachádzajú aj metódy nedefinované žiadnou z tried v súbore. Jedná sa o názvy metód definovaných v systémových knižniciach.

Triedy

Definícia triedy obsahuje typ, prístupový príznak, typ nadtriedy, rozhranie triedy a niekoľko dalších údajov. Záznam `class_data_offset` určuje pozíciu dátovej položky v rámci datovej sekcie. Každý takýto záznam obsahuje štyri zoznamy identifikujúce triedou definované polia a metódy. Identifikátor metódy obsahuje aj offset štruktúry `code_item`, udržujúcej samotný bajtkód a mnoho ďalších informácií potrebných pre exekúciu alebo ladenie.

Ďalšie dáta

Z prílohy A je zrejmé, že formát obsahuje radu ďalších, v tejto práci vynechaných, sekcií a záznamov. Jedná sa napríklad o anotácie, dáta pre staticky linkované súbory a množstvo záznamov opomenutých pri popise základných štruktúr.

Kapitola 4

Zmeny vo formáte LOFF

Pri návrhu spôsobu konverzie obecne používaných formátov do formátu LOFF a počas práce na programe vykonávajúcom tieto konverzie vzniklo niekoľko návrhov na zmeny a rozšírenia v internom formáte projektu. Niektoré z nich boli prijaté, zahrnuté do špecifikácie formátu a implementované do knižnice *objfilelib* predstavenej v podkapitole 5.1. Je nutné poznamenať, že zmeny popísané v tejto kapitole nie sú všetky, ktoré boli vykonané na formáte, ale len tie navrhnuté a implementované autorom práce. Príloha A poskytuje zjednodušený pohľad na štruktúru originálneho formátu a jeho modifikácií.

4.1 Adresovanie po bajtoch

Najväčšia zmena zasahujúca takmer všetky časti formátu sa týka spôsobu adresovania. V podkapitole 3.1 sa uvádza, že v čase začiatku práce na projekte bola veľkosť dát v sekcii, a všetky offsety do týchto dát uložené v slovách. Z toho vyplýva, že ak bola napríklad veľkosť slova zdrojového súboru tridsaťdva bitov a jeden bajt mal osem bitov, musela byť veľkosť všetkých sekcií a tiež všetky offsety do sekcií násobkom štyroch. V opačnom prípade nastal problém s uložením hodnoty do LOFF formátu. Na tento prístup sa však nedá vždy spoliehať. V skutočnosti ani najpoužívanejšie procesory nevyžadujú takto striktné zarovnanie a prekladač môže vygenerovať súbor, ktorý sa nebude dať presne preložiť. Príklady 4.1.1 a 4.1.2 (jedná sa o výstupy programu *readelf*) ilustrujú niektoré z problémov, ktoré nastali pri konverzii testovacieho ELF súboru, preloženého prekladačom *gcc* verzie *4.6.0* na tridsaťdva bitovom OS Linux pre architektúru *i386*.

Príklad 4.1.1 Nepresná veľkosť sekcie

```
Section Headers:
   [Nr] Name              Type     Addr     Off  Size ES Flg Lk Inf Al
   [ 0]                   NULL     00000000 0000 0000 00      0   0  0
   [ 1] .interp           PROGBITS 08048114 0114 0013 00   A~0   0  1
   [ 2] .note.ABI-tag NOTE     08048128 0128 0020 00   A~0   0  4
   ...

  0x13  (veľkosť sekcie .interp v~bajtoch)
* 0x08  (bitov v~bajte)
------
  0x98 = 152 bitov
   152 / 32 (dĺžka slova) = 4.75 veľkosť sekcie v~slovách
```

Príklad 4.1.2 Nepresný offset symbolu

```
Symbol table '.symtab' contains 96 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx  Name
   ...
    13: 08048710     0 SECTION LOCAL  DEFAULT   13
   ...
    35: 080487a0     0 FUNC    LOCAL  DEFAULT   13  symbol#1
   ...
    82: 08048897   115 FUNC    GLOBAL DEFAULT   13  symbol#2
   ...

  0x080487a0  (symbol#1)
- 0x08048710  (sekcia 13)
------------
        0x90 = 144 bajtov = 1152 bitov
        1152 / 32 (dĺžka slova) = 36 slovo v~LOFF sekcií

  0x08048897  (symbol#2)
- 0x08048710  (sekcia 13)
------------
       0x187 = 391 bajtov = 3128 bitov
        3128 / 32 (dĺžka slova) = 97.75 slovo v~LOFF sekcií
```

Riešením je upustiť od adresavania po celých slovách a začať adresovať po bajtoch. Zmena ovplyvnila údaje o veľkosti sekcie, dáta sekcie (je možné, že posledné slovo – riadok, dát nie je úplné), offsety relokácií a symbolov do dát a tiež veľkosť relokačného bloku.

4.2 Zaznamenanie vstupnej adresy programu

Vstupná adresa programu (*entry point*) je adresa, od ktorej sa začne vykonávať program po jeho načítaní do pamäte. Tento údaj nie je nutný v aplikáciach určených pre vstavané zariadenia, keďže väčšina z nich má pevne danú adresu kde sa nahrá kód, a ktorá bude vykonaná po reštarte. Programy určené pre exekúciu na stroji s operačným systémom sa však bez určenia vstupu väčšinou nezaobídu.

Do hlavičky formátu bola pridaná dvojica záznamov. Prvý určuje, či je vstupný bod nastavený (hodnota 1), alebo nenastavený (hodnota 0) a druhý predstavuje samotnú hodnotu vstupnej adresy.

4.3 Rozšírenie príznakov

Na dvoch miestach boli do formátu pridané nové príznaky. V hlavičke sekcie pribudli:

- *debu**G*** – typ určený na označenie sekcií obsahujúcich ladiace informácie [4].
- ***I**nfo* – tento typ je určený pre sekcie nenáčítané do pamäte pred vykonaním programu. Problém pôvodných príznakov bol ten, že reprezentovali iba sekcie do pamäte načítané. Konverziou do LOFFu sa stali všetky sekcie načítateľné a po konverzií späť do originálneho formátu im tento príznak ostal. Pri pokuse o spustenie programu však nastala chyba, keďže niektoré sekcie boli v pamäti navyše.

Typy symbolov boli rozšírené o:

- ***A****bsolute* – špecialny typ symbolu, väčšinou sa používa pre uchovanie názvov súborov.
- ***C****ommon* – symbol može byť definovaný niekoľkokrát.

4.4 Zmena datového typu masky relokácie

V pôvodnej implementácií knižnice objfilelib boli všetky číselné hodnoty uložené v rozsahu celého čísla veľkosti tridsaťdva bitov. Existujú však položky, ktorým tento rozsah nestačí. Jednou z nich je maska relokácie. Jej hodnota može byť až takého rozsahu, akého je pole vymedzené relokáciou. Z tohto dôvodu bolo nutné zmeniť rozsah masky až na šesťdesiatštyri bitov. Zmena si vyžiadala kompletnú reimplementáciu triedy zapúzdrujúcej prácu s celými číslami v knižnici objfilelib. Použitím šablón, funkcií schopných pracovať so šesťdesiatštyri bitovými číslami a pokročilými triedami C++ pre manipuláciu reťazcov bola dosiahnutá schopnosť triedy spracovať celé čísla ľubovoľného rozsahu a znamienka.

V súčasnosti je maska relokácie jediný záznam využívajúci tieto nové schopnosti, úprava však v budúcnosti dovolí väčšiu špecializáciu dátových typov aj u iných položiek.

Kapitola 5

Nástroje pre vzájomnú konverziu formátov

Konverzie medzi objektovými formátmi sa väčšinou využívajú v prípadoch reverzného inžinierstva a transformácií programov napísaných pre zastaralé architektúry v prípade, že nie sú dostupné ich zdrojové kódy alebo je z nejakého dôvodu ich rekompilácia nemožná. Niekoľko projektov ktoré v minulosti úspešne využili konverziu formátov:

- *Cygnus GNU-Win32 Project* – poskytuje porty GNU nástrojov pre OS Windows.
- *Projekt DIAMONDS* – má za cieľ navrhnúť nový 8/16 bitový *RISC* mikrokontrolér a zaistiť programovú kompatibilitu s existujúcou rodinou *CISC*.
- *Macintosh Application Environment* – program umožňoval užívateľom niektorých UNIX pracovných staníc spúšťať *Apple Macintosh* aplikácie.
- *Wabi* – program firmy *Sun* prekladal objektové súbory *MS Windows 3.x* pre spustenie na OS *Solaris*.

Cieľom všetkých vyššie uvedených projektov je, aby bol výsledný program úspešne spustiteľný na inej architektúre procesoru, prípadne inom OS. Za týmto účelom nestačí len zmeniť objektový formát, je nutná konverzia inštrukcií procesoru a/alebo systémových volaní. To však v projekte Lissom nie je nutné a konverzia prebieha tak, že vstup je načítaný do interných štruktúr vhodnej knižnice. Táto reprezentácia je namapovaná do štruktúr výstupneho formátu tak, aby sa stratilo čo najmenej údajov. Ďalšia vhodná knižnica vytvorí výstupný súbor a využitím funkcií oboch knižnic sa dáta prekopírujú.

5.1 Objfilelib

Knižnica *objfilelib*[1] je súčasťou nástrojov projektu Lissom, kde zaisťuje manipuláciu s objektovým formátom LOFF. Pôvodnú verziu vytvoril v roku 2006 Libor Vašíček, tá bola naďalej rozširovaná podľa potrieb projektu. Zmeny formátu LOFF navrhnuté v kapitole 4 boli implementované práve v tejto knižnici.

[1] *Objfilelib* – názov knižnice v rámci projektu Lissom. Neexistuje žiadny oficiálny názov.

5.2 Knižnica BFD

Binary File Descriptor (BFD) [3], je knižnica vyvinutá spoločnosťou *Cygnus Support*, napísaná v jazyku C a šírená pod licenciou *GNU General Public License*. Úlohou BFD je poskytnúť aplikáciam mechanizmy pre jednotnú manipuláciu s veľkým množstvom objektových formátov. Tieto služby využívajú napríklad programy ako *GNU Assembler, GNU Linker, GNU Debugger* a ďalšie. Knižnica v súčasnosti podporuje niekoľko desiatok objektových formátov a architektúr procesorov.

Podľa [3] môžeme BFD rozdeliť na dve časti:

- *Front-end* – poskytuje užívateľom rozhranie, spravuje pamäť a kanonické dátové štruktúry. Tiež rozhoduje, ktorý back-end použiť a kedy volať jeho funkcie.
- *Back-end* – poskytuje BFD spojenie s reálnym svetom (konkrétne formáty). Každý back-end definuje množinu funkcií, ktoré môže front-end použiť na správu svojej kanonickej podoby. Back-end môže pre svoju potrebu udržiavať ďalšie informácie za účelom väčšej efektívnosti.

Vyššie zmienené kanonické štruktúry predstavujú internú reprezentáciu kde BFD využitím správneho back-endu načíta objektový súbor. Jedná sa vlastne o úplne nový formát navrhnutý tak, aby dokázal bez straty pojať informácie z ľubovoľného objektu. Pomocou funkcií poskytovaných front-endom je možné s touto reprezentáciou pohodlne pracovať aj bez hlbšej znalosti pôvodného formátu.

Na začiatku podkapitoly je uvedené, že BFD podporuje desiatky formátov. Drvivá väčšina z nich ale nie je zahrnutá v štandardnej knižnici získanej ako binárny balíček z repozitára, alebo preloženej zo zdrojových súborov s východzím nastavením prekladu. Tieto knižnice v skutočnosti obsahujú iba povinné back-endy a back-endy odvodené od architektúry stroja kde sa prekladalo. Z tohto dôvodu boli zdrojové súbory BFD pripojené k projektu Lissom a sú prekladané špeciálnym zostavovacím súborom (*Makefile*). Ten prinúti BFD Makefile zahrnúť do výsledku nasledujúce formáty a architektúry:

- Architektúra *ARM* pre formáty ELF a PE.
- Architektúra *i386* pre formáty ELF, PE a Mach-O[2]
- Architektúra *x86_64* pre formáty ELF a PE.
- Celá škála procesorov *MIPS* asociovaných s formátom ELF.

5.3 Knižnica PeLib

PeLib [1] je open source knižnica napísaná v jazyku C++ uľahčujúca modifikáciu súborov vo formáte Portable Executable. Knižnica dokáže načítať objektový súbor do tried reprezentujúcich jeho štruktúry a poskytuje metódy pre jednoduchú manipuláciu týchto informácií. Pomocou PeLib je možné vykonať prakticky ľubovoľné zmeny v existujúcom PE súbore, prípadne úplne od základov vytvoriť nový súbor. V tomto prípade je ale knižnica využitá len na malú úpravu hlavičky súboru vytvoreného pomocou BFD.

[2] *Mach-O* (Mach object) je formát používaný operačnými systémami *NeXTSTEP*, *Darwin* a *Mac OS X*. Transformácia formátu nie je súčasťou tejto práce, ale je plánovaná v blízkej budúcnosti.

5.4 Knižnica E32lib

Keďže BFD, a ani žiadna iná dostupná knižnica, nie je schopná práce s E32Image formátom, bolo nutné vytvoriť pre potreby konvertoru knižnicu vlastnú. Nesie názov *E32lib* a je distribuovaná pod licenciou projektu Lissom. E32lib využíva služby knižnice *deflate* (súčasť *Symbian SDK*) šírenej pod *Symbian Example Source Code* licenciou. Deflate poskytuje funkcie pre komprimáciu a dekomprimáciu E32Image súboru.

V prílohe B je znázornený zjednodušený model tried použitých v E32lib pre internú reprezentáciu formátu. Model sa zameriava na zobrazenie rozdelenia jednotlivých sekcií E32Image na logické celky (triedy) a informácie v nich uložené. Schéma zanedbáva všetky metódy tried a tiež niektoré vzťahy dedičnosti.

Z modelu je zrejmé, že E32Image súbor ako celok reprezentuje trieda `E32File`, ktorej atribútmi sú ukazovatele na ostatné časti súboru. Táto hlavná trieda implementuje radu metód rozdeliteľných do nasledujúcich kategórií:

- Metóda `loadFile()` načíta súbor do vnútornej reprezentácie. Pred samotným načítaním `checkInputFile()` skontroluje, či je vstup spracovateľný[3] a v prípade, že je komprimovaný, metóda `uncompress()` využije knižnicu deflate na dekomprimáciu. V tejto fáze je ďalšie načítanie delegované na `load()` funkcie jednotlivých zložiek hlavnej triedy.

- Metóda `saveFile()` uloží vnútornú reprezentáciu do súboru. Po vytvorení výstupu je samotné ukladanie opäť prenechané na `save()` funkcie podsekcií.

- Metóda `fixImage()` by mala byť zavolaná užívateľským programom vždy pred použitím `saveFile()`. Volaním funkcií `fix()` všetkých podsekcií sa zaistí opravenie prípadných nezrovnalostí. Tie mohli vzniknúť napríklad rozširovaním existujúceho súboru. Volanie nie je vynútené funkciou `saveFile()` pre prípad, že užívateľský program vie čo robí a praje si vytvoriť súbor s istým druhom chyby.

- Metódy `has*()` informujú o prítomnosti sekcií v súbore.

- Metódy `get*()` vracajú ukazovateľ na niektorú zo sekcií.

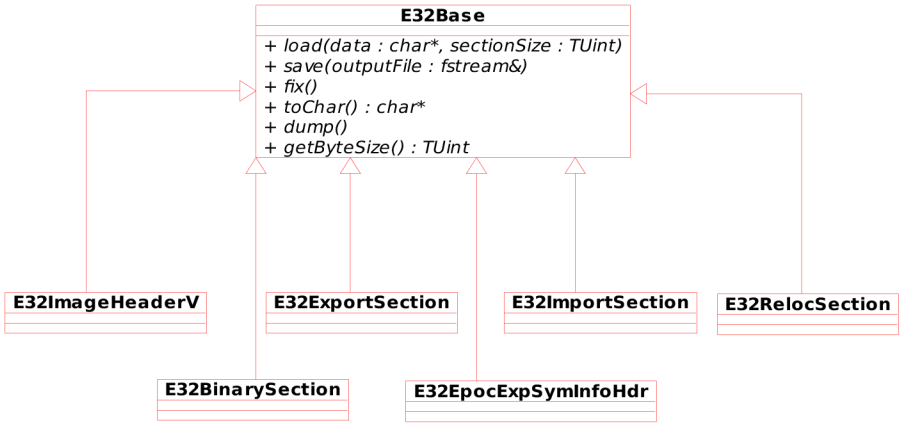
- Metódy `dump*()` tlačia obsah sekcií na štandardný výstup. Delegujú na `dump()` podsekcií.

Obrázok 5.1 zobrazuje model dedenia čisto virtuálnej triedy `E32Base` a funkcie, ktoré musia odvodené triedy implementovať. Okrem už zmienených metód `load()`, `save()`, `fix()`, `dump()`, je zdedená aj `toChar()` ktorá vráti ukazovateľ na pole bajtov s binárnou reprezentáciou obsahu štruktúr. Veľkosť poľa je zhodná s návratovou hodnotou `getByteSize()`.

Atribúty jednoduchých dátových typov sú vo všetkých triedach verejné. Nemá zmysel pre každý takýto atribút definovať dvojicu `get*()`, `set*()` metód. Výnimkou sú niektoré hodnoty veľkostí sekcií priamo odvodené z dát sekcie. Všetky atribúty zložených typov sú privátne a triedy poskytujú funkcie na ich manipuláciu. Ďalej platí, že ak sú prvky zoznamu zloženého dátového typu, zoznam udržuje len ukazovateľe na ne.

Knižnica E32lib dokáže načítať E32Image súbor do internej reprezentácie, ponúka metódy na jej manipuláciu a spätné uloženie do súboru. Pomocou knižnice je dokonca možné

[3] Musí sa jednať o E32Image súbor. Na základe príznakov v hlavičke sa určí či je knižnica schopná tento konkrétny súbor spracovať.



Obrázok 5.1: Vzťah dedičnosti medzi základnou triedou a sekciami E32Image súboru.

od základu vytvoriť nový súbor. Aj keď bola knižnica vytvorená pre potreby konverzie medzi E32Image a LOFF formátom, vďaka vyššie popísaným princípom a schopnostiam je ju možné použiť pre ľubovoľnú prácu s objektovým formátom používaným OS Symbian.

5.5 Knižnica DEXlib

DEXlib[4] je knižnica napísaná v jazyku C, distrubuovaná pod *Apache License verzie 2.0*. Knižnica je súčasťou *Android SDK* s ktorým je úzko spätá a pôvodne bol jej preklad možný len s využitím zostavovacieho systému Android. Pre oddelené použitie v rámci projektu Lissom bolo nutné vykonať nasledujúce zmeny:

- Základ tvoria zdrojové súbory z SDK[5] v podadresáry `dalvik/libdex`.
- Ďalej bolo nutné pridať `dalvik/vm/Common.h` a `dalvik/vm/DalvikVersion.h`.
- Knižnica využíva modul `safe_iop`[6] implementujúci funkcie pre bezpečné operácie s celými číslami.
- Prípony všetkých zdrojových súborov boli zmenené z `*.c` na `*.cpp` a na preklad knižnice sa používa prekladač jazyka C++.
- Zmena prekladača si vyžiadala na mnohých miestach doplniť explicitné pretypovanie.
- V knižnici bola na niekoľkých miestach upravená práca s ukazovateľmi. Pôvodná verzia sa spoliehala, že veľkosť každého ukazovateľa bude štyri bajty, čo spôsobilo chyby na šesťdesiatštyri-bitových systémoch.
- Pôvodná knižnica využívala logovacie makrá z hlavičkového súboru `Log.h`, ktoré boli späté s ďalším kódom SDK. Bol vytvorený nový hlavičkový súbor s makrami vypisujúcimi logovacie správy na štandardný výstup.

[4] Názov knižnice v Android SDK je *libdex*. Názov bol zmenený na DEXlib aby sa meno knižnice nelíšilo od ostatných mien v projekte Lissom, kde je *lib* ako postfix.

[5] http://code.google.com/p/pdn-slatedroid/source/browse/trunk/eclair/

[6] http://code.google.com/p/safe-iop/

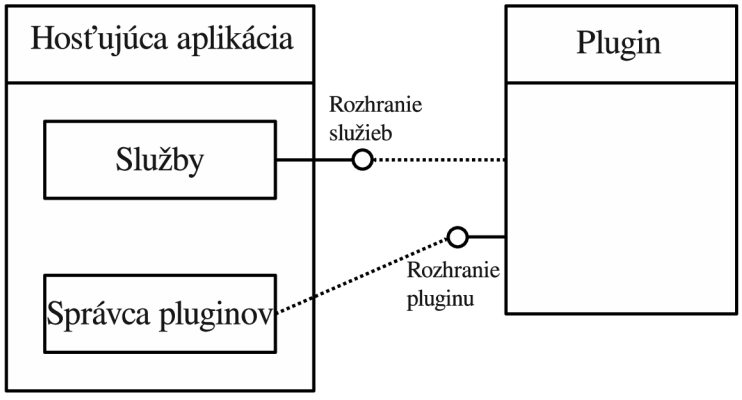
Kapitola 6

Aplikácia bintran

Výsledkom práce je konzolová aplikácia nazvaná *bintran*, ktorej úlohou je obojsmerná konverzia medzi obecne používanými formátmi (kapitola 3) a interným formátom projektu Lissom. Aplikácia je implementovaná v jazyku C++ a je navrhnutá ako *systém zásuvných modulov* (pluginovací systém). Jednotlivé časti sú popísané ďalej v tejto kapitole.

6.1 Pluginovacie systémy

Pluginy sú vhodné pre vývoj dynamických systémov, pretože umožnujú rozširovanie systému vývojármi tretích strán. Verzia aplikácie vyvinutá autorom práce obsahuje niekoľko pluginov implementujúcich základné konverzie, ale každý užívateľ môže vytvoriť vlastný plugin implementujúci špecifickú konverziu bez toho, aby mal k dispozicii zdrojový kód pôvodnej aplikácie alebo ostatných pluginov. Jediné, čo vývojár potrebuje, je poznať definíciu rozhrania pluginu očakávanú hosťujúcou aplikáciou. Aplikácia bintran je na tento prístup mimoriadne vhodná, keďže každá konverzia z jedného formátu do druhého je prakticky samostatný program, nezávislý a nespolupracujúci so svojim okolím.



Obrázok 6.1: Model spolupráce hosťujúcej aplikácie a pluginu.

Obrázok 6.1 znázorňuje zjednodušený model spolupráce hosťujúcej aplikácie a pluginu.

6.2 Rozhranie aplikácie v C/C++

Jazyk C++ nie je uspôsobený pre jednoduchý vývoj pluginov. Štandard jazyka nešpecifikuje ABI (*Application Binary Interface*), čo spôsobuje nekompatibilitu knižníc preložených rôznymi prekladačmi, alebo dokonca rôznymi verziami rovnakého prekladača. C++ neobsahuje ani jednotný koncept načítania dynamických knižníc a tak každá platforma (operačný systém) ponúka vlastné riešenie problému.

Výhodou implementácie systému pluginov v C++ je len väčšia prehľadnosť a čistota kódu vďaka využitiu virtuálnych tried pre definíciu rozhrania. Všeobecne sa však odporúča využiť na rozhranie jazyk C, ktorý je v praxi kompatibilný medzi všetkými prekladačmi. Toto riešenie je použité aj aplikáciou bintran a jej pluginmi. Nad triedami implementujúcimi jednotlivé konverzie je vystavané rozhranie napísané v čistom C.

Samotné rozhranie je definované v samostatnom hlavičkovom súbore, ktorého zdrojový kód bude prístupný vývojárom tretích strán. Jedná sa o veľmi jednoduché rozhranie uchovávajúce o plugine minimum informácií a vyžadujúce definíciu len dvoch funkcií.

Informácie o plugine sú obsiahnuté v štruktúre `SPluginInfo` zobrazenej v zdrojovom kóde 6.1.

```
1 struct SPluginInfo {
2     const char *pluginName;              // Meno pluginu.
3     const char *pluginAuthor;            // Autor pluginu.
4     const char *pluginVersion;           // Verzia pluginu.
5     const char *pluginUsage;             // Popis pouzitie pluginu.
6     int  pluginTransformationCount;      // Počet transformácií pluginu.
7     const char **pluginTransformations; // Zoznam mien transformácií.
8 };
```

Zdrojový kód 6.1: Štruktúra obsahujúca informácie o plugine.

Zdrojový kód 6.2 zobrazuje štruktúru popisujúcu rozhranie (`SPluginInterface`) využívanú správcom pluginov v hosťujúcej aplikácií. Externé deklarácie jednotlivých položiek štruktúry zaistia, že budú definované každým modulom, ktorý vloží tento hlavičkový súbor. To znamená, že každý plugin je prinútený obsahovať štruktúru, ktorá ho popisuje, funkciu, ktorá ho inicializuje a funkciu, ktorá vykoná transformáciu nastavenú pri inicializácií.

```
1 struct SPluginInterface {
2     int (*initPlugin)(const char *, const char *, int, int, char**);
3     int (*convertByPlugin)();
4     struct SPluginInfo info;
5 };
6
7 extern struct SPluginInfo pluginInfo;
8 extern "C" int initPlugin(const char *input, const char *output, int
      transformTypeIdx, int optionCount, char **options);
9 extern "C" int convertByPlugin();
```

Zdrojový kód 6.2: Štruktúra rozhrania pluginu.

Registrácia pluginu do správy pluginov je zjednodušená funkciou `getBintranPlugin()` (zdrojový kód 6.3), definovanou priamo v hlavičkovom súbore, aby sa predišlo chybám pri jej implementácií. Funkcia vytvorí štruktúru popisujúcu plugin, naplní ju informáciami modulu, ktorý vložil hlavičkový súbor a vráti ukazovateľ na túto štruktúru. Všetko, čo musí správca pluginov spraviť, je nájsť v každej knižnici túto funkciu, zavolať ju a uložiť návratovú hodnotu.

```
1 extern "C" struct SPluginInterface *getBintranPlugin() {
2     struct SPluginInterface *pi = (struct SPluginInterface*)
3         malloc (sizeof (struct SPluginInterface));
4     pi->initPlugin = &initPlugin;
5     pi->convertByPlugin = &convertByPlugin;
6     pi->info = pluginInfo;
7
8     return pi;
9 }
```

Zdrojový kód 6.3: Funkcia pre registráciu pluginu v hostiteľskej aplikácií.

6.3 Hostiteľská aplikácia

Hostiteľská aplikácia (bintran) je jednoduchý program, úlohou ktorého je spracovať vstupné parametre, dynamicky načítať špecifikované knižnice, vybrať z nich tú, ktorá implementuje požadovanú konverziu a spustiť jej vykonanie.

Spracovanie argumentov príkazovej riadky je vykonané za pomoci knižnice `getopt`. Parametre programu s krátkym popisom zobrazuje tabuľka 6.1. Parametre `-p` a `-s` slúžia pre načítanie pluginov. Možnosť `-t` určuje typ konverzie vstupného súboru do výstupného.

Mená pluginov sa uložia do zoznamu. V prípade, že bol zadaný adresár, uložia sa všetky podsúbory s názvom obsahujúcim podreťazec **.so* alebo **.dll*. V ďalšom kroku program prechádza zoznam mien a pokúša sa otvoriť špecifikované knižnice. Ako už bolo spomenuté na začiatku kapitoly, tento proces je závislý na operačnom systéme. Pod OS Linux sa použije funkcia `dlopen()`, pod OS Windows funkcia `LoadLibrary()`. Obe vrátia takzvaný handler knižnice nutný pre ďalšiu prácu. Nasleduje nájdenie symbolu (funkcie) v otvorenej knižnici. Program hľadá len jednu funkciu, `getBintranPlugin()`. Jej následným volaním získa štruktúru popisujúcu rozhranie pluginu. Hľadanie symbolu zabezpečujú funkcie `dlsym()` (Linux) a `GetProcAddress()` (Windows). Po ukončení práce bintran zatvorí všetky úspešne otvorené knižnice pomocou funkcií `dlclose()` (Linux) a `FreeLibrary()` (Windows). Je nutné poznamenať, že v prípade ak, niektorá zo spomenutých funkcií skončí s chybou, program neukončí svoju činnosť. Označí knižnicu ako nenačítanú a pokračuje spracovaním ďalšieho súboru. Rozhodovanie medzi linuxovou a windowsovou variantou funkcií sa deje pomocou podmieneného prekladu s využitím makra `_WIN32`.

Po načítaní knižníc, program prechádza zoznam rozhraní, kým nenájde plugin implementujúci typ konverzie zadaný parametrom `-t`. Nájdený plugin je následne inicializovaný vstupným a výstupným súborom a nerozpoznanými parametrami hostiteľského programu. Ak inicializácia prebehne bez chyby je zavolaná funkcia `convertByPlugin()`.

Prepínač	Popis
`-i <file>` `--input=<file>`	Vstupný súbor.
`-o <file>` `--output=<file>`	Výstupný súbor.
`-p <file>` `--plugin-path=<file>`	Plugin alebo adresár s pluginmi.
`-s <file>` `--plugin-source=<file>`	Súbor obsahujúci cesty k pluginom.
`-t <type>` `--transformation-type=<type>`	Typ transformácie.
`-h` `--help`	Tlač nápovedy.

Tabuľka 6.1: Parametre programu bintran.

6.4 Plugin bfdtoloff

Plugin *bfdtoloff* slúži na konverziu formátov ELF a PE do formátu LOFF. Za týmto účelom využíva knižnice BFD a objfilelib. Celú konverziu zapúzdruje trieda `CBfdToLoff` inicializovaná funkciou rozhrania pluginu `initPlugin()`. Plugin má len jeden parameter slúžiaci na výpis zoznamu architektúr a formátov, ktoré je schopná spracovať knižnica BFD (tabuľka 6.2).

Prepínač	Popis
`-l` `--arch-target-list`	Zoznam architektúr a formátov podporovaných používanou verziou BFD.

Tabuľka 6.2: Parametre pluginu bfdtoloff.

6.4.1 Postup konverzie

Vstupný súbor je načítaný do internej BFD reprezentácie[1], knižnica objfilelib vytvorí výstupný súbor a nasleduje konverzia rozdeliteľná na nasledujúce kroky:

- *Nastavenie architektúry* – výber relevantných informácií z BFD hlavičky a ich uloženie do LOFF hlavičky. Informácie, ktoré LOFF nepojme sa stratia.
- *Konverzia sekcií* – aj keď vo formátoch ELF a PE sú prakticky všetky dáta sekcie, v BFD reprezentácií sú tabuľky symbolov a relokácie uložené v osobitných štruktúrach. Táto akcia teda získa len sekcie v pravom slova zmysle. Ako prvá je pre každú sekciu vytvorená hlavička s menom, typom a adresou v pamäti. Následne sú

[1] Pri použití funkcie `bfd_openr()` väčšinou nie je nutná špecifikácia formátu a architektúry vstupu. Knižnica dokáže tieto informácie odvodiť z hlavičky súboru. Explicitné zadanie hodnôt je potrebné len v prípade, ak používaná verzia BFD knižnice obsahuje niekoľko back-endov schopných pracovať s formátom. Tento prípad však v súčasnosti používanej verzii nenastane. Aj napriek tomu plugin implementuje algoritmus pre riešenie takejto situáciu.

skopírované dáta sekcie. Všetky ostatné položky hlavičky vyplní knižnica objfilelib automaticky na základe obsahu LOFF súboru.

- *Konverzia symbolov* – ku každému BFD symbolu je vytvorený LOFF symbol s rovnakým menom, odpovedajúcim typom, sekciou a adresou symbolu. Pred uložením symbolu do LOFF reprezentácie sa program uistí, že ukladaný symbol má jedinečné meno.

6.4.2 Rozšírenia pluginu bfdtoloff

Prvotnou úlohou práce bola konverzia spustiteľných súborov obecne sa skladajúcich zo sekcií a tabuľky symbolov (tá tiež nie je nevyhnutná). Počas práce však vyvstala potreba konverzie objektových súborov a knižníc. Preto musel byť plugin bfdtoloff rozšírený o algoritmus schopný otvoriť a spracovať archívy (skladajú sa z niekoľkých objektových súborov) a konverziu relokácií.

Program ešte raz prejde všetky sekcie a v prípade, že pre sekciu existujú relokácie (príznak `SEC_RELOC`) získa ich popis. Relokácie v podaní BFD sú veľmi zložité štruktúry. Sémantika mnohých položiek je závislá na type relokácie, tých existujú stovky[2] a formát LOFF ich nie je schopný zaznamenať. Transformované relokačné záznamy sú v súčasnosti použité len na vymaskovanie bitov, ktoré by boli v prípade linkovania súboru zmenené. Vykonanie relokácií len na základe informácií vo formáte LOFF je bez rozšírenia formátu nemožné.

6.4.3 Zhodnotenie konverzie

Formáty ELF, PE a interná reprezentácia BFD sú omnoho bohatšie ako formát LOFF. Počas konverzie sa nenávratne stratí mnoho informácií z hlavičky súboru, hlavičiek sekcií a relokačných záznamov. Aj napriek tomu dokáže LOFF zaznamenať všetky skutočne kritické informácie spustiteľného súboru, čo dokazuje kapitola 7. Konverzia objektových súborov v súčasnosti nemá za úlohu vytvoriť relokovateľné objekty formátu LOFF, plní len pomocnú funkciu pri dekompilácií. V budúcnosti by však aj bez rozšírenia formátu bolo možné pridať konverziu dynamickej tabuľy symbolov (`.dynsym`) a dynamických relokácií. Tie aktuálna verzia programu preloží len ako obyčajné binárne sekcie.

6.5 Plugin lofftobfd

Plugin *lofftobfd* vykonáva opačnú konverziu k pluginu bfdtoloff. Vstup je načítaný do reprezentácie objfilelib a knižnica BFD sa postará o vytvorenie a naplnenie výstupného súboru. V tomto prípade je ale nutná konkrétna špecifikácia formátu a architektúry výstupu. Zo vstupného LOFF súboru je preto definovaných celkovo štrnásť typov konverzií do rôznych kombinácií cieľov (ELF, PE) a architektúr procesorov. Parametre pluginu (tabuľka 6.3) medzi iným tiež umožnujú definovať konkrétny typ procesoru (nutné hlavne pre MIPS).

6.5.1 Postup konverzie

Proces transformácie je pomerne jednoduchý, keďže odpadajú problémy prekladu bohatšieho formátu do jednoduhšieho. V BFD reprezentácií sa nastavia hodnoty hlavičky, vytvoria

[2] Typ relokácie určuje spôsob aplikácie hôdnot ako maska, posunutie, offset, atď pri výpočte skutočnej adresy pri relokačnom procese.

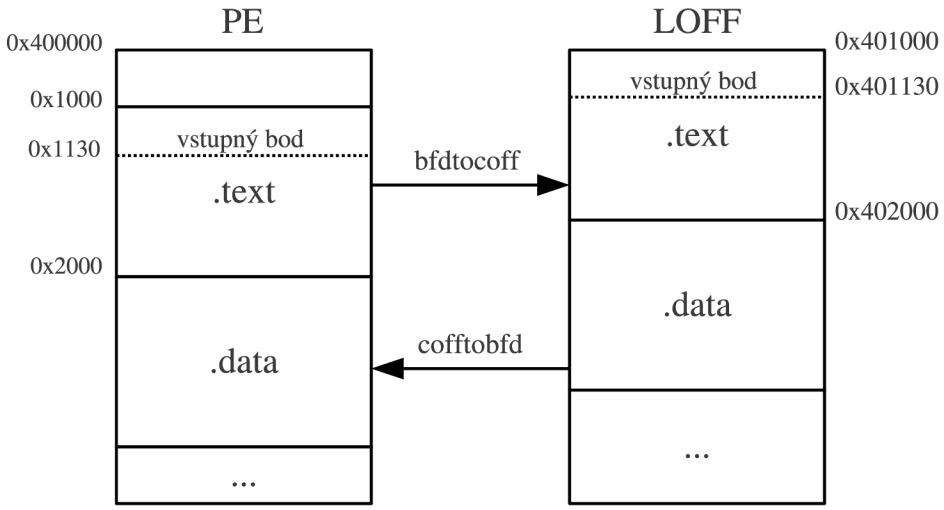
Prepínač	Popis
`-a <type>` `--arch-type=<type>`	Architektúra výstupného súboru.
`-b <x>` `--image-base=<x>`	Bázová adresa výstupného súboru. Primárne pre PE. Východzia hodnota je `0x400000`.
`-g <x>` `--section-alignment=<x>`	Zarovnanie sekcií výstupu. Primárne pre PE. Východzia hodnota je `9`.
`-u <type>` `--subsystem=<type>`	Subsystém výstupného PE súboru. LEN pre PE.
`-m <type>` `--major-subsystem-version=<type>`	Verzia subsystému PE súboru. LEN pre PE.
`-l` `--arch-target-list`	Zoznam architektúr a formátov podporovaných používanou verziou BFD.

Tabuľka 6.3: Parametre pluginu lofftobfd.

hlavičky sekcií, naplní tabuľka symbolov a nakoniec skopírujú dáta sekcií. Tento plugin nepodporuje konverziu objektov alebo knižníc, a preto sú prípadné relokácie ignorované.

6.5.2 Úprava výsledných PE súborov

Konverzia do formátu PE síce vytvorí podľa špecifikácie korektný súbor, ale bez doplnenia špecifických informácií nebude na systéme Windows spustiteľný. Najväčší problém nastáva s hodnotou vstupnej adresy programu. Operačný systém Windows očakáva, že každý spustiteľný program bude načítaný do pamäte na adresu `0x400000` (takzvaný *Image Base*) a že adresy sekcií v pamäti a tiež vstupná adresa programu budú od tohto bodu relatívne. BFD ani LOFF nie sú schopné tento prístup zaznamenať a tak sú všetky dotknuté hodnoty zvýšené o Image Base (obrázok 6.2). Riešením je Image Base pri prevode do PE odčítať. Práve na tento účel slúži parameter `-b`.



Obrázok 6.2: Rozdiely adresavania medzi formátmi PE a LOFF.

LOFF neudržuje informáciu o zarovnaní sekcií. Tento nedostatok sa nijak neprejavil u testovaných ELF súborov, ale spôsobuje problémy u PE (rozdielne očakávania OS). Informácia može byť v tomto prípade zadaná parametrom `-g`. Posledné dve možnosti pluginu slúžia pre špecifikáciu podsystémov OS Windows.

Jednoduchú nápravu vyššie zmienených nedostatkov zabezpečuje knižnica PeLib.

6.6 Plugin e32toloff

Plugin transformuje súbor vo formáte E32Image do formátu projektu Lissom. Konverziu zapúzdruje trieda `CE32ToLoff` a pre spracovanie vstupu je využitá knižnica E32lib. E32toloff neprijíma žiadne parametre.

6.6.1 Postup konverzie

V podkapitole 3.4.2 bolo uvedené že všetky E32Image súbory sú určené pre tridsaťdva bitovú little-endian architektúru. Z tohto dôvodu je vyplnie hlavičky LOFF veľmi jednoduché. V ďalšom kroku sú skopírované sekcie v poradí: textová, neinicializované data, exportované symboly, pomenované exportované symboly, importované symboly a relokácie. Vzniknuté sekcie vo formáte LOFF nesú preddefinované, vždy rovnaké názvy. Odpovedajúce príznaky a ich dáta sú presnou kópiou obsahu pôvodných sekcií.

E32Image neobsahuje štandartnú tabuľku symbolov, avšak po jeho načítaní do štruktúr E32lib je možný prístup k tabuľkám exportov. Obe tabuľky (jednoduchá aj pomenovaná) obsahujú v podstate rovnaké záznamy, druhá z nich však nesie dotatočné informácie (hlavne mená symbolov). Z tohto dôvodu sa konvertuje vždy len jedna z nich, pričom prioritu majú práve pomenované exporty. Ak neexistujú, každému symbolu sa priradí meno zložené s `exportSymbol@@` a poradového čísla exportu. Je jasné že všetky symboly sú verejného typu a ich adresa ukazuje do textovej sekcie.

Podobná extrakcia prebieha aj u importovaných symbolov. Adresa, typ a sekcia sú opäť zrejmé, ostáva určiť meno. To je zložené z troch položiek:

- Meno knižnice, v ktorej je symbol definovaný.
- Ordinálne číslo symbolu v knižnici. Táto šestnásťbitová hodnota sa nachádza v slove určenom adresou importu v textovej sekcií. Ďalších šestnásť bitov predstavuje takzvaný *adjustment*, ten je v súčasnosti ignorovaný.
- Je možné, aby bol jeden symbol z knižnice vložený na niekoľko miest v `.text` sekcií. Potom na každé takéto miesto bude ukazovať jedna položka tabuľky importov. Ak by sa teda meno symbolu skladalo len z predchádzajúcich dvoch hodnôt, vznikli by vo výstupnom LOFF súbore rovnomenné symboly. To ale špecifikácia formátu nedovoľuje a je nutné mená nejak odlíšiť. Z toho dôvodu je na koniec pridané číslo určujúce koľkokrát bol už tento symbol importovaný.

Zo vstupu by bolo ďalej možné získať relokácie a vytvoriť pre ne príslušné záznamy. Ako už bolo spomenuté, jedná sa len o takzvané fix-up relokácie ktorých konverzia zatiaľ nie je potrebná.

6.7 Plugin lofftoe32

Jedná sa o plugin vykonávajúci opačnú konverziu k e32toloff. Konverziu zapúzdruje trieda `CLoffToE32` a výstupný súbor je vytvorený pomocou knižnice E32lib. Kritické informácie nezaznamenané formátom LOFF sa dajú nastaviť parametrami pluginu, tabuľka 6.4.

Prepínač	**Popis**
`-r <x>` `--process-priority=<x>`	Priorita výslednej aplikácie. Východzia hodnota je `foreground`.
`-c <x>` `--cpu=<x>`	Architektúra výslednej aplikácie. Východzia hodnota je `armV5`.

Tabuľka 6.4: Parametre pluginu lofftoe32.

6.7.1 Postup konverzie

Ide o konverziu skladajúcu sa s prakticky totožných funkcií, s ktorých každa preloží jednu sekciu. Ich názov musí byť rovnaký ako nastavil e32toloff. Jediná povinná sekcia je `.text`, ostatné nemusia byť prítomné. Funkcie sú od seba oddelené z dôvodu prípadných budúcich rozšírení. Transformácia zanedbáva tabuľku symbolov aj relokácie, práve toto by sa dalo zmeniť tak, aby plugin dokázal vytvoriť sekcie importov a exportov len na základe symbolov vo vstupnom LOFF.

Po naplnení štruktúr E32lib je zavolaná metóda knižnice `fixImage()`, tá opraví adresy v hlavičke podľa aktuálneho obsahu.

6.8 Plugin dextoloff

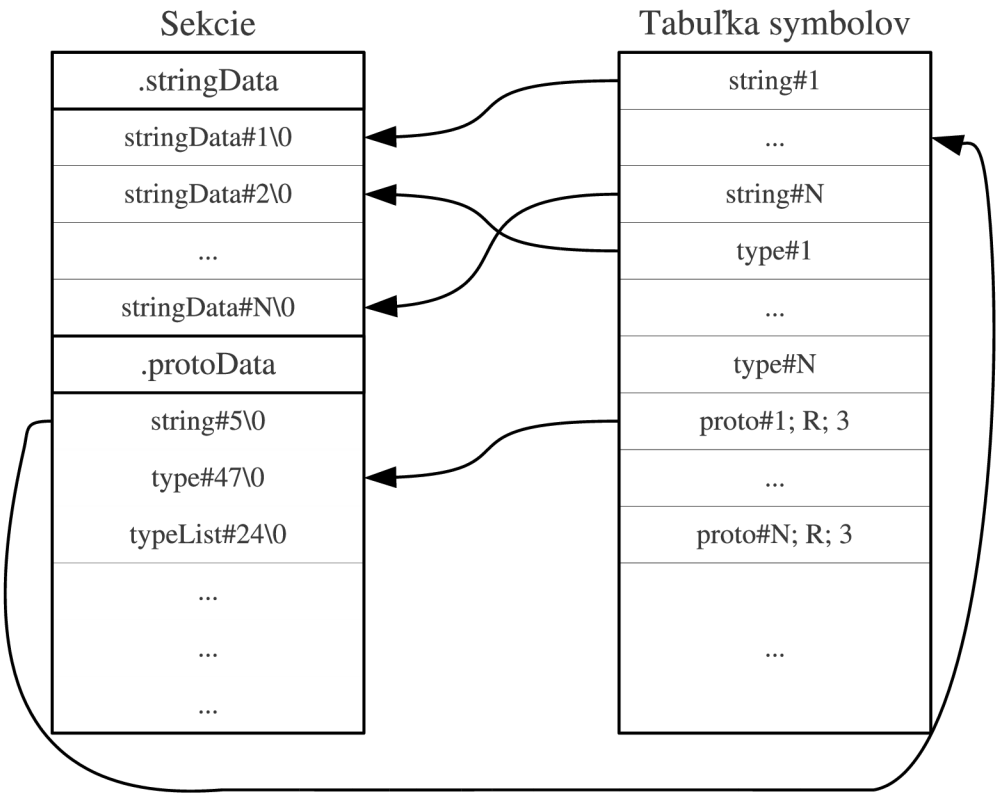
Cieľom pluginu *dextoloff* je konvertovať bajtkód popísaný v podkapitole 3.5 do objektového formátu LOFF. Keďže sú oba formáty veľmi rozdielne, jedná sa o veľmi náročnú úlohu. Súbory DEX sú prakticky kolekciou tried určených pre interpretáciu virtuálnym strojom, zatiaľ čo LOFF predstavuje klasický objektový formát, aké vznikajú prekladom zdrojového kódu. Cieľom konverzie nieje nutne preložiť celý DEX, stačí samotný bajtkód inštrukcií a dostatočné množstvo metadat na to, aby bol výsledok dekompilovateľný.

6.8.1 Postup konverzie

Hlavným rozdielom, s ktorým sa treba vysporiadať je fakt, že DEX sa skladá zo štruktúr, ktorých položky často odkazujú do iných štruktúr. Sémantika takýchto hodnôt sa však nedá odvodiť z dát v súbore – je určená špecifikáciou formátu. Na obrázku v prílohe A.2 napríklad vidíme, že jedna definícia prototypu (`Proto_ids`) sa skladá z troch hodnôt. Prvá z nich určuje index do poľa reťazcov, druhá index do poľa typov a tretia zoznam parametrov. Význam hodnôt je nám síce známy, zistili sme ho ale zo špecifikácie, nie z položky samotnej. Jednoduchým preložením takýchto dát do formátu LOFF by vznikol súbor, ktorého sekcie by síce bez straty obsahovali všetky povôdné informácie, ale k ich správnej interpretácií by sme sa nepriblížili.

Návrh riešenia je zobrazený na obrázku 6.3. Okrem klasických symbolov (na obrázku `string#*` a `type#*`) ukazujúcich na sekciu so skutočnými dátami existujú aj špeciálne

symboly (na obrázku `proto#*`). Tie obsahujú rozširujúcu informáciu (v tomto prípade R; 3), tá vraví, že na adrese symbolu sa nachádzajú tri referencie. Každá referencia je meno symbolu ukazujúceho na dáta, v pôvodnom DEX súbore identifikované indexom. Týmto sa čiastočne vyriešil najväčší problém. Program spracúvajúci takýto LOFF už nemusí vedieť že v sekcií `.proto_data` sú položky zložené z troch indexov do sekcií `string`, `type` a `typeList`. Táto informácia je teraz obsiahnutá v samotnom LOFF a stačí sledovať symboly, kým nenarazí na taký, ktorý identifikuje dáta a nie referenciu.



Obrázok 6.3: Spôsob uloženia odkazov vo formáte LOFF.

S vyššie popísaným mechanizmom môžeme pristúpiť k samotnej konverzií. Hlavička DEX formátu okrem offsetov a veľkostí obsahuje iba informáciu o spôsobe uloženia (takmer vždy little-endian), výstupu sú preto priradené štandardné hodnoty – tridsaťdvabitové slovo zložené zo štyroch bajtov.

Už z obrázku 6.3 je zrejmé, že dáta reťazcov sú uložené do sekcie `.stringData` a pre každú položku je vytvorený symbol `string#*`. Pôvodná sekcia `Type_ids` je len zoznam indexov na reťazce a preto nie je nutné vytvárať samostatnú sekciu v LOFF. Namiesto toho vzniknú symboly `type#*`, ich index odpovedá poradiu v zozname `Type_ids` a ich adresa identifikuje reťazec v sekcií `.stringData`.

Pre konverziu sekcie `Type_list` však už treba využiť referencie. Všetky zoznamy typov sú uložené ako referencie na symboly typov v sekcií `.typeListData`. Pre každý zoznam existuje symbol `typeList#*` typu referencia udávajúci adresu začiatku v sekcií a počet odkazov ktoré na nej nájdeme.

Spôsob uloženia `Proto_ids` bol už popísaný na začiatku tejto podkapitoly a je využitý

aj pre sekciu `Field_ids`.

Hlavným cieľom je konverzia samotných bajtkódov metód. Program prechádza definície tried a každej vytvorí samostatnú sekciu. Tá nesie meno triedy a ukazuje na ňu symbol `class#*` s indexom, aký má trieda v zozname definícií. Program ďalej prečíta offset dát triedy a v tejto štruktúre nájde informácie o metódach. Bajtkódy metód sú skopírované do sekcie triedy, na ich začiatok sú opäť vytvorené symboly `method#*`. Nakoniec prejde sekciu `Method_ids`, pre všetky doteraz netransformované metódy spraví externé symboly (jedná sa o systémove metódy).

Takto vytvorený LOFF súbor by mal obsahovať dostatok informácií pre základnú dekompiláciu. Plugin sa však bude v budúcnosti nepochybne ďalej vyvíjať tak, aby spĺňal potreby dekompilátoru.

Kapitola 7

Testovanie aplikácie

V tejto kapitole sú popísané testy aplikácie bintran a jej pluginov. Ich účelom je preukázať funkčnosť a použiteľnosť programu v rôznych situáciach. Aplikácia, knižnice, testované súbory a skripty pre automatizované testovanie sa nachádzajú na priloženom CD, viz C.

7.1 Testovanie konverzie formátov ELF a PE do formátu LOFF

Cieľom testu je preukázať, že konverzia obecne používaných formátov do formátu LOFF zachová dostatočné množstvo informácií pre úspešné spustenie konvertovaného programu.

7.1.1 Princíp testu

Zdrojové kódy vybraných programov sú preložené na spustiteľné aplikácie vo formátoch ELF a PE. Tieto programy sú konvertované pomocou pluginu bfdtoloff do formátu LOFF. Následne je použitý plugin lofftobfd pre opätovné vytvorenie spustiteľných súborov v pôvodných formátoch. Samotný test spočíva v spustení originálneho a vygenerovaného programu s rovnakým vstupom a porovnanie výstupov.

7.1.2 Špecifikácia testu

Testovanie bolo vykonané na architektúre procesoru *i386* a operačných systémoch Windows a Linux. Na oboch systémoch boli konvertované vybrané programy z projektu *MiBench*[1] poskytujúceho aplikácie určené pre testovanie vstavaných systémov. Jedná sa o konzolové programy napísané v jazyku C prevažne implementujúce obecne používané algoritmy (*crc*, *quicksort*, *dijkstra*, *sha*, atď). Pre každý OS bolo konvertovaných aj niekoľko špecifických aplikácií ako napríklad program pracujúci s procesmi OS Linux alebo aplikácia s grafickým užívateľským rozhraním pre OS Windows.

7.1.3 Výsledok testu

Testovanie MiBench aplikácií odhalilo stratu kritického príznaku sekcií `SEC_THREAD_LOCAL` indikujúceho operačnému systému, že daná sekcia musí byť asociovaná individuálne s každým vláknom – za behu sa vytvorí toľko kópií sekcie, koľko je vlákien. Takéto sekcie sa do spustiteľných aplikácií dostali po tom, čo boli zdrojové súbory preložené s príznakom

[1] *http://www.eecs.umich.edu/mibench/*

prekladu `-static`. Ten spôsobil, že sa do objektových súborov vložil potrebný kód zo všetkých knižníc, ktoré program využíval a ktoré by boli inak linkované dynamicky za behu aplikácie. V súčasnej dobe nie je prioritou projektu Lissom pracovať s aplikáciami využívajúcimi vlákna a tento príznak nateraz nebude do špecifikácie formátu LOFF pridaný. Ďalšie testovanie prebehlo bez statického linkovania.

Pre úspešný beh aplikácie sú najkritickejšie správne hlavičky sekcií, tabuľka symbolov ani relokácie nie sú potrebné[2]. Príklad 7.1.1 porovnáva údaje o sekciach pôvodného a transformovaného ELF súboru. Červenou farbou sú označené zmenené hodnoty. Vidíme, že sa jedná o zarovnanie sekcií a tiež že konverzia pridá k príznaku `A` (*alloc*) príznak `W` (*write*). Aj napriek týmto zmenám je väčšina vygenerovaných programov úspešne spustiteľná na oboch operačných systémoch.

Problémy nastali len vo formáte PE v dvoch situáciách:

- Pri konverzií aplikácií využívajúcich veľké statické polia. Tento problém je pravdepodobne spôsobený stratou informácie určenej pre operačný systém o veľkosti rezervovanej pamäti na zásobníku (`SizeOfStackReserve`). Je možné, že sa prejaví aj u rezervovanej pamäti na hromade (`SizeOfHeapReserve`).
- Pri konverzií aplikácie *putty* s grafickým uživateľským rozhraním (ostatné GUI programy boli preložené bez problémov). Je pravdepodobné, že chyba nastala po tom, čo konverzia nenaplnila všetky kritické položky rozšírenej PE hlavičky (hlavne rôzne informácie o podsystémoch OS).

Transformácie týchto aplikácií budú podrobená ďalším testom.

Príklad 7.1.1 Porovnanie sekcií súborov.

```
[Nr] Name              Type      Addr     Off  Size ES Flg Lk Inf Al
...
[10] .init             PROGBITS  0804857c 057c 0017 00  AX  0   0  4(1)
[11] .plt              PROGBITS  08048594 0594 0150 04  AX  0   0  4(1)
[12] .text             PROGBITS  080486f0 06f0 0604 00  AX  0   0 16(1)
[13] .fini             PROGBITS  08048cf4 0cf4 001c 00  AX  0   0  4(1)
[14] .rodata           PROGBITS  08048d10 0d10 013f 00  WA  0   0  4(1)
[15] .eh_frame_hdr     PROGBITS  08048e50 0e50 001c 00  WA  0   0  4(1)
[16] .eh_frame         PROGBITS  08048e6c 0e6c 0058 00  WA  0   0  4(1)
[17] .ctors            PROGBITS  08049000 1000 0008 00  WA  0   0  4(1)
...
```

7.2 Testovanie konverzie formátu LOFF do formátu ELF

Cieľom testu je preukázať funkčnosť konverzie formátu projektu Lissom do obecne používaných formátov.

[2]Dynamické linkovanie používa dynamickú tabuľku symbolov a dynamické relokácie, tie sú však preložené ako obyčajné datové sekcie a tak sa zachovajú v originálnej podobe.

7.2.1 Princíp testu

V rámci projektu Lissom vzniklo niekoľko modelov procesorov na základe ktorých je možné automaticky vygenerovať nástoje pre danú architektúru. Idea testu je využiť vzniknutý prekladač na vytvorenie takých programov vo formáte LOFF, ktorých výsledok exekúcie je dopredu známy. Tieto súbory sú pluginmi lofftobfd a bfdtoloff skonvertované do obecne používaných formátov a naspäť. Originálny LOFF súbor a výsledok konverzie sú následne spustené v simulátore daného CPU a výstup je porovnaný s očakávaným výsledkom.

7.2.2 Špecifikácia testu

Ako testovacie programy boli opäť použité upravené aplikácie projektu MiBench. Generovanie nástrojov prebieha pre achitektúru MIPS[3] (big-endian) a LOFF súbory sú transformované do formátu big-endian ELF pre procesor MIPS (je dôležité zachovať rovnaký spôsob uloženia v pamäti).

7.2.3 Výsledok testu

Všetky testované programy boli úspešne transformované a odsimulované. Keďže je formát ELF bohatší ako LOFF, prakticky nedošlo ku strate žiadnych informácií (okrem údajov o číslach riadkov, ktoré sa nekonvertujú).

7.3 Testovanie konverzie medzi formátmi E32Image a LOFF

Cieľom testu je overiť funkčnosť pluginov e32toloff, lofftoe32 a knižnice E32lib.

7.3.1 Princíp testu

Opäť sa jedná o dvojitú konverziu vstupných E32Image súborov do LOFF súborov a naspäť. V tomto prípade však nebolo dostupné zariadenie, na ktorom by bolo možné spustiť výsledok a tak bolo nutné spoliehať sa na porovnanie výstupov programov *elf2e32* (súčasť Symbian SDK) a *e32dump* (pomocný program vytvorený autorom práce). Nejedná sa ale o vážny nedostatok, pretože konverzie nestratia žiadne kritické informácie a výstupný program je takmer totožný s originálom, čo dokazuje nasledujúca podkapitola.

7.3.2 Špecifikácia testu

Testovacie programy vznikli prekladom demonštračných aplikácií obsiahnutých v *Symbian SDK S60 5th Edition* pomocou prekladača ktorý SDK ponúka. Kedže súbory E32Image formátu možu byť niekoľkých rôznych podtypov, aj testovanie prebehlo na niekoľkých sadách vstupov:

- *EXEXP* – spustiteľné EXE súbory obsahujúce tabuľku exportovaných symbolov.
- *STDEXE* – spustiteľné EXE súbory obsahujúce tabuľku pomenovaných exportovaných symbolov.
- *DLL* – knižnice, ktoré sa ale od spustiteľných súborov prakticky nelíšia.

[3] Autor práce nie je v tomto prípade autorom úprav testovacích programov ani modelu architektúry MIPS.

7.3.3 Výsledok testu

Vďaka princípu prekladu zachovávajúceho binárne dáta sekcií v nedotknutom stave a pretože formát E32Image je veľmi jednoduchý sú vygenerované výstupy prakticky totožné so vstupmi. Jediné miesto z ktorého sa môžu stratiť informácie je hlavička.

Príklad 7.4.1 porovnáva hlavičky typického spustiteľného súboru a výsledku konverzie. Modrou sú označené položky ktoré konverzia síce zmení, ale ich nový obsah je korektný a odpovedá novej hlavičke. Patria sem napríklad údaje o čase vytvorenia súboru alebo checksumi, tie samozrejme musia byť prepočítané. Červené sú informácie konverziou stratené. Jedná sa o:

- *Uid2* – slúži na odlíšenie aplikácií s rovnakými Uid1. Nejedná sa o kritickú informáciu.
- *Uid3* – unikátne pre každú aplikáciu. Výrobca softwaru musí požiadať o pridelenie od spoločnosti Nokia. Položka by v budúcnosti mohla byť nastaviteľná pomocou parametru pluginu.
- *HeapSizeMin, HeapSizeMax, StackSize* – Informácie o pamäti potrebnej pre exekúciu aplikácie. Je pravdepodobné že, v prípade neuvedenia pridelí OS aplikácií východzie hodnoty.

V podkapitole 3.4 je uvedené, že moderné E32Image súbory majú ďalšie dve hlavičky. Prvá z nich však nesie len údaj o veľkosti súboru po dekompresií a druhá informácie o bezpečnosti programu (identifikácia výrobcu, atď). Ani jedna z nich nie je nevyhnutne nutná a ich hodnoty sú často nevyplnené.

7.4 Testovanie konverzie z formátu DEX do formátu LOFF

Cieľom testu je demonštrovať schopnosť pluginu dextoloff transformovať niektoré informácie z DEX súborov do formátu LOFF. Kedže nie sú konvertované všetky potrebné dáta pre beh aplikácie a neexistuje plugin schopný opačnej konverzie, model virtuálneho stroja ani funkčný dekompilátor nie je v tejto fáze možné testovať správnosť procesu spustením výsledného súboru. Jediná možná verifikácia konverzie je manuálne porovnať výsledok prekladu s výstupom programu *dexdump* – dokáže vytlačiť informácie o súbore vo formáte DEX.

Príklad 7.4.1 Porovnanie hlavičiek súborov.

```
Uid1              = 0x1000007a
Uid2              = 0x20004c45
Uid3              = 0xe80000c6
UidChecksum       = 0xdf14adb5
Signature         = 0x434f5045
HeaderCrc         = 0xb96196b7
ModuleVersion     = 0xa0000
CompressionType   = 0
TimeLo            = 0x2e9b7a00
TimeHi            = 0x00e17b7c
Flags             = 0x1200042a
CodeSize          = 0x6878
DataSize          = 0
HeapSizeMin       = 0x1000
HeapSizeMax       = 0x100000
StackSize         = 0x5000
BssSize           = 0
EntryPoint        = 0x3700
CodeBase          = 0x8000
DataBase          = 0x400000
DllRefTableCount  = 0x13
ExportDirOffset   = 0x6748
ExportDirCount    = 0xa
TextSize          = 0x6878
CodeOffset        = 0x9c
DataOffset        = 0
ImportOffset      = 0x6914
CodeRelocOffset   = 0x7688
DataRelocOffset   = 0
ProcessPriority   = 0x15e
CpuIdentifier     = 0x2001
```

Kapitola 8

Záver

Cieľom bakalárskej práce bolo priblížiť tematiku konverzie formátov spustiteľných súborov. Za týmto účelom boli predstavené obecne používané formáty ELF, PE, E32Image, DEX a tiež formát LOFF projektu Lissom. Ďalej boli spomenuté niektoré projekty zaoberajúce sa podobnými konverziami a knižnice použité za týmto účelom.

Významnou časťou celej práce bol návrh a implementácia vlastnej knižnice poskytujúcej mechanizmy pre jednoduchú manipuláciu s formátom E32Image. Táto úloha bola o to ťažšia, že spoločnosť Nokia nezverejnila špecifikáciu svojho formátu. Bolo nutné zdĺhavé štúdium zdrojových súborov vybraných aplikácií Symbian SDK a použitie reverzného inžinierstva na analýzu E32Image súborov. Výsledná knižnica nesie názov E32lib a významne uľahčila konverziu formátu operačného systému Symbian. Jedná sa však o komplexný nástroj, ktorý može nájsť využitie prakticky pri akéjkoľvek práci s týmto formátom.

Pred samotnou implementáciou konvertoru bol nutný návrh spôsobu transformácie. Jedná sa hlavne o podrobné pochopenie syntaxe a sémantiky všetkých vyššie uvedených formátov a následný spôsob namapovania informácií medzi obecne používanými formátmi a formátom LOFF. Počas tejto práce došlo k odhaleniu niekoľkých nedostatkov formátu LOFF, následnému návrhu riešení a ich implementácií. Tá si vyžadovala oboznámenie sa s existujúcim kódom projektu Lissom manipulujúcim s interným formátom.

Výsledkom je konzolová aplikácia bintran napísaná v jazyku C++. Je navrhnutá ako pluginovací systém, čo umožní jednoduchý vývoj modulov, nezávisle na už existujúcich zdrojových kódoch. Samotné konverzie sú zapúzdrené práve v oddelených pluginoch, dynamicky načítaných pomocou rozhrania definovaného v špecialnom hlavičkovom súbore. V rámci práce vznikli tieto pluginy:

- *Bfdtoloff* – implemetuje konverzie formátov spracovateľných knižnicou BFD do formátu LOFF.
- *Lofftobfd* – opačné konverzie k bfdtoloff.
- *E32toloff* – implemetuje konverziu formátu E32Image (a jeho podtypov) do formátu LOFF.
- *Lofftoe32* – opačná konverzia k e32toloff.
- *Dextoloff* – implemetuje konverziu formátu DEX do formátu LOFF.

Funkčnosť všetkých pluginov bola otestovaná sadou testou. Zvýšená pozornosť bola venovaná testovaniu konverzií najrozšírenejších formátov ELF a PE. Veľkou výhodou je,

že správnosť ich konverzie sa dá overiť spustením transformovaných súborov na bežnom osobnom počítači.

Ďalšia práca na projekte bude mať charakter implementácie nových pluginov podporujúcich ďalšie formáty, prípadne rozšírenie pluginov už existujúcich podľa potrieb projektu. Najväčší potencial na rozšírenia majú nepochybne moduly bfdtoloff a lofftobfd, keďže knižnica BFD dokáže pracovať s desiatkami ďalších formátov. Táto vlastnosť bude v krátkom čase využitá pre konverziu formátu Mach-O, používaného operačnými systémami NeXTSTEP, Darwin a Mac OS X. Takmer určite bude treba rozšíriť a upraviť konverziu dextoloff a reagovať na prípadné chyby odhalené v ostatných pluginoch.

Ako bolo uvedené na začiatku tejto práce, hlavným využitím vytvoreného programu je predspracovanie vstupov obecnému dekompilátoru, prípadne testovanie binárnej kompatibility výstupov ostatných nástrojov projetu Lissom. Aplikácia sa teda podieľa na takých úlohách ako je súbežný návrh technického a programového vybavenia, alebo vírová analýza, ktorých praktické využitie je nepopierateľné.

Literatura

[1] PeLib [online]. http://www.pelib.com/about.php.

[2] E32Image, Nokia forum wiki [online].
http://wiki.forum.nokia.com/index.php/E32Image, 2007-12-08 [cit. 2010-1-6].

[3] Chamberlain, S.: *The Binary File Descriptor Library*. Iuniverse Inc, September 2000, ISBN 978-0595136193.

[4] DWARF Debugging Information Committee: *DWARF Debugging Information Format*. Čtvrté vydání, June 2010.
URL http://www.dwarfstd.org

[5] Harrison Richard, M. S.: *Symbian OS C++ for Mobile Phones volume 3*. Wiley, 2007, ISBN 0-470-85611-4.

[6] Kolář, D., Husár, A.: *Návrh výstupního formátu pro assembler a linker*. Interní dokumentace. Brno, FIT VUT v Brně, 2010.

[7] Křoustek, J.: *Analýza a převod kódu do vyššího programovacího jazyka*. diplomová práce, Brno, FIT VUT v Brně, 2009.

[8] Levine, J. R.: *Linkers and Loaders*. Morgan Kaufmann Publishers, Inc., 1999, ISBN 1-55860-496-0.

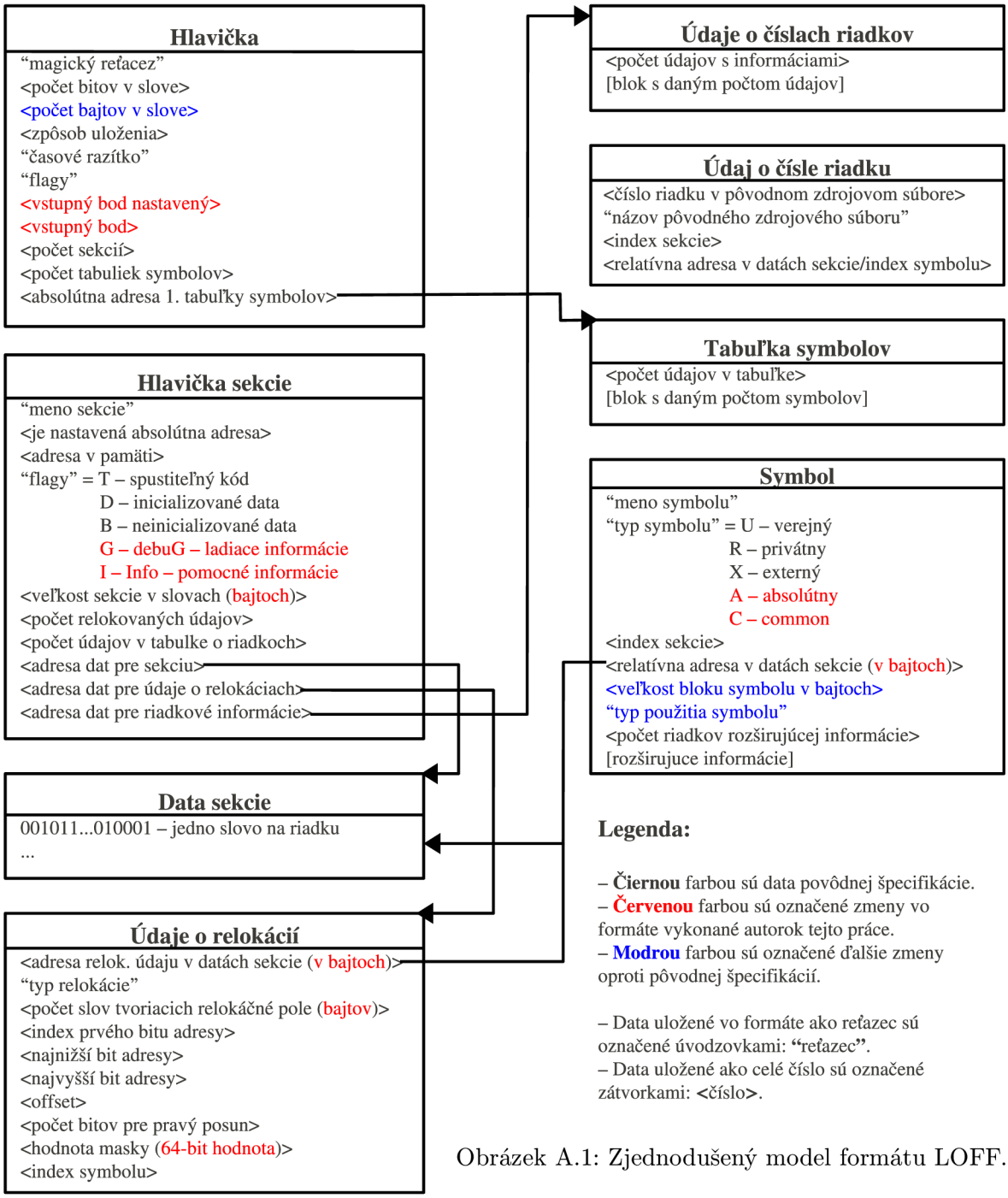
[9] Masařík, K.: *Systém pro souběžný návrh technického a programového vybavení počítačů*. VUTIUM, Faculty of Information Technology BUT, první vydání, 2008, ISBN 978-80-214-3863-7, 156 s.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8915

[10] Microsoft Corporation: *Microsoft Portable Executable and Common Object File Format Specification*. 8 vydání, February 2008.
URL http://msdn.microsoft.com/en-us/windows/hardware/gg463119.aspx

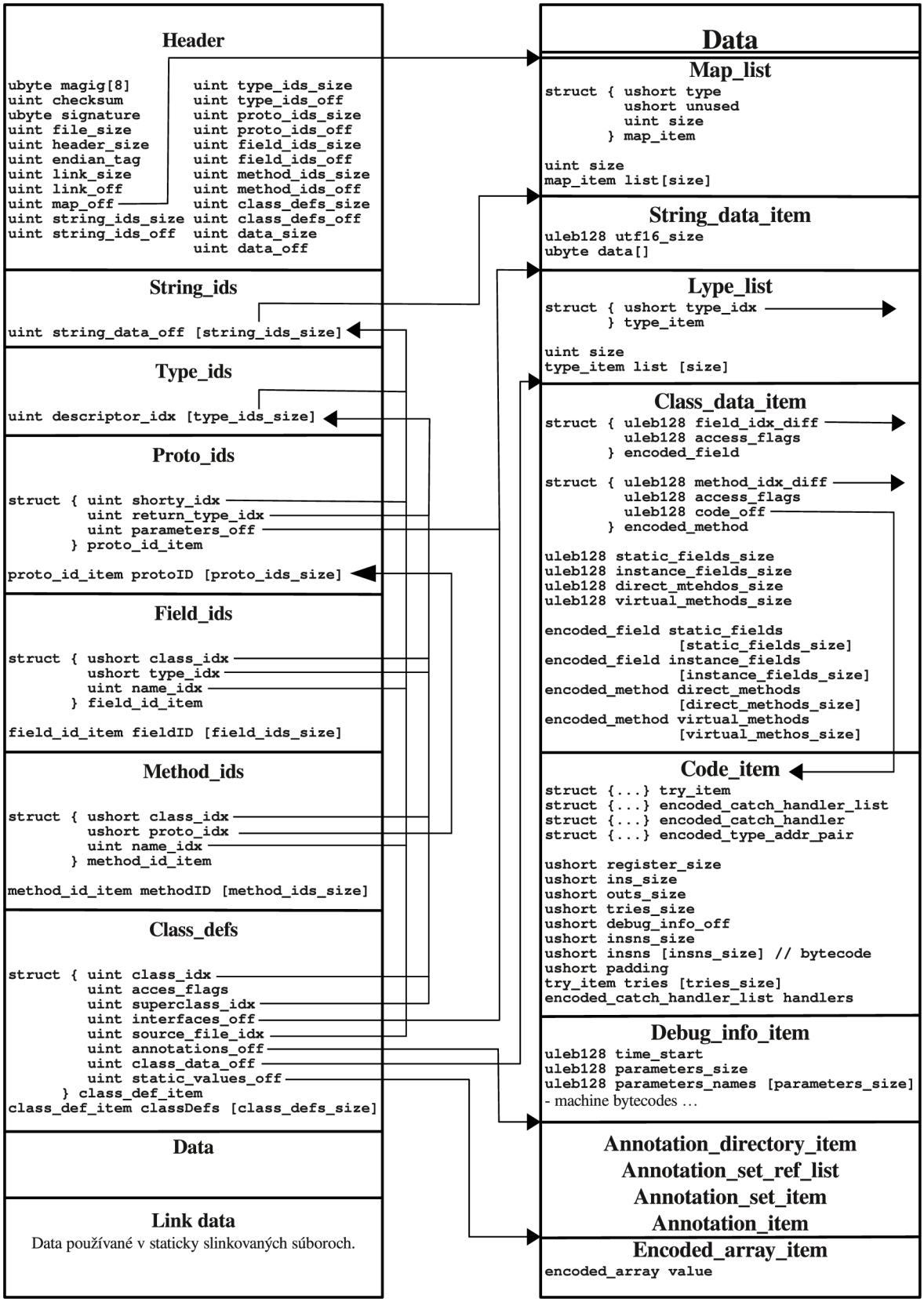
[11] TIS Committee: *Executable and Linking Format Specification*. První vydání, May 1995.

Příloha A

Model formátov LOFF a DEX



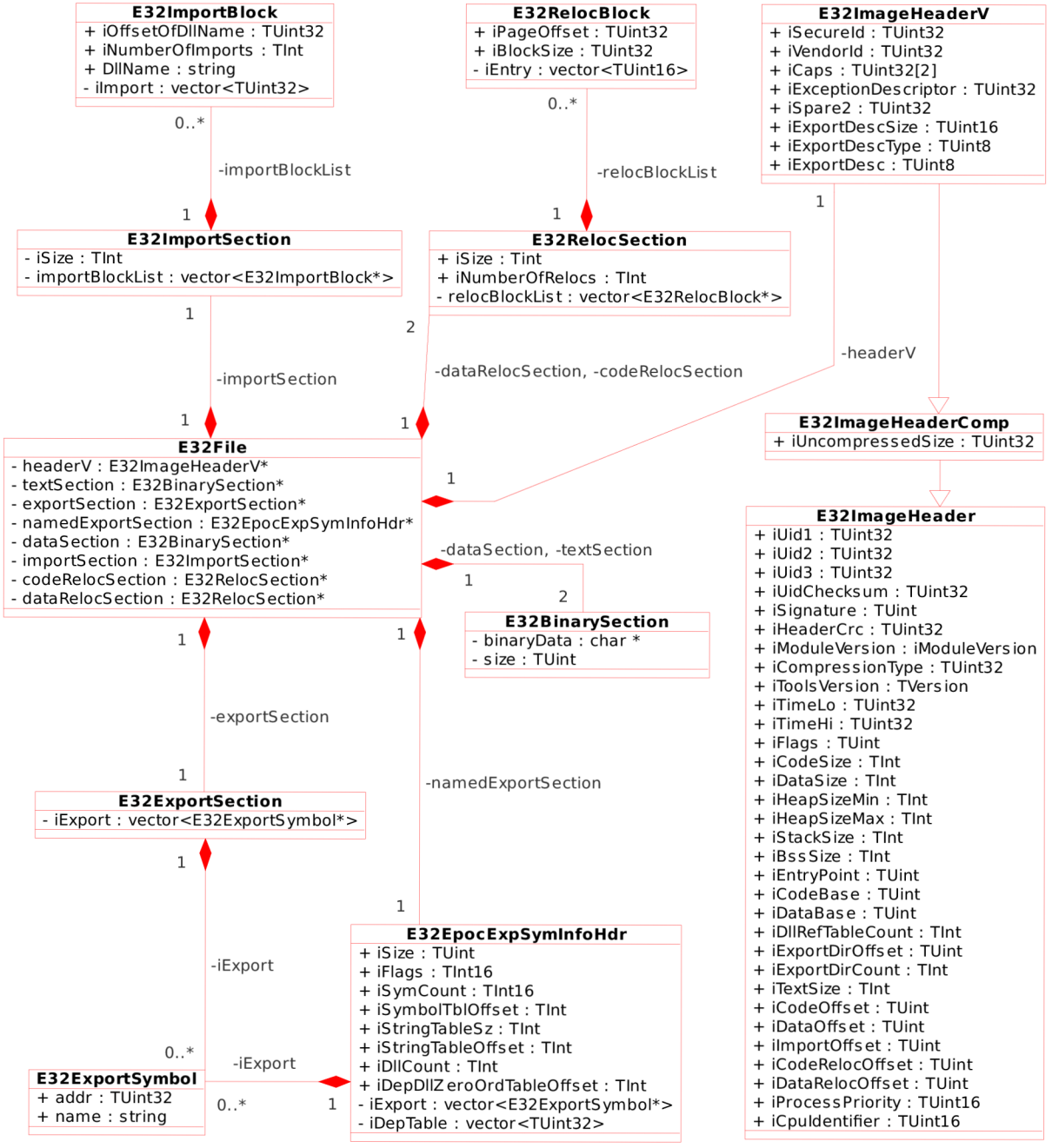
Obrázek A.1: Zjednodušený model formátu LOFF.



Obrázok A.2: Zjednodušený model formátu DEX.

Příloha B

Model tried E32lib.



Příloha C

Obsah CD

Adresár	**Popis**
./Makefile	Súbor pre automatizovaný preklad.
./README.txt	Informácie o CD.
./bintran/	Zdrojový kód aplikácie bintran.
./bintran/plugins/	Zdrojový kód pluginov.
./doc/	Elektronická verzia tohto dokumentu.
./documentation/	Doxygen dokumentacia k programom.
./lissom_headers/	Hlavičkové súbory k Lissom knižniciam.
./lissom_libs/	Preložené knižnice projektu Lissom využívané programom bintran.
./object_format_libs/	Zdrojové kódy knižníc manipulujúcich s objektovými formátmi.
./tests/	Testovacie programy a skript na ich spustenie.