VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

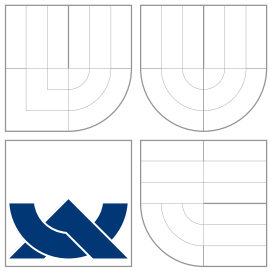
FRAMEWORK PRO SESTAVENÍ A TESTOVÁNÍ BEZPEČNOSTNÍHO SÍŤOVÉHO ŘEŠENÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

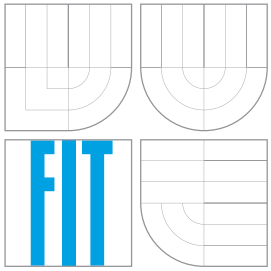
AUTOR PRÁCE
AUTHOR

JIŘÍ SUŠKA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

FRAMEWORK PRO SESTAVENÍ A TESTOVÁNÍ BEZPEČNOSTNÍHO SÍŤOVÉHO ŘEŠENÍ

FRAMEWORK FOR BUILDING AND TESTING NETWORK SECURITY SOLUTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — JIŘÍ SUŠKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MATES VOJTĚCH
SUPERVISOR

BRNO 2011

Sem prosím vložte zadání

Abstrakt

Tato diplomová práce se zabývá problematikou automatického sestavování a testování bezpečnostního síťového řešení AVG for Linux/FreeBSD na platformách GNU/Linux a FreeBSD. Popisuje co je a k čemu slouží AVG for Linux/FreeBSD. Překlad a sestavování jsou rozebrány od zdrojových kódů po distribuční balíčky, které si uživatelé mohou nainstalovat na svůj počítač. V práci je také zmíněno co jsou repozitáře, jakým způsobem je možné je vytvořit a k čemu se používají. Část věnující se testování AVG for Linux/FreeBSD se zabývá návrhem vhodných automatických testů tohoto produktu a jejich implementací. V rámci praktické části byl navržen a implementován testovací nástroj a s jeho pomocí bylo AVG for Linux/FreeBSD otestováno sadou implementovaných testů.

Abstract

This master thesis discourses upon problems with automatic building and testing of network security solution AVG for Linux/FreeBSD on platforms GNU/Linux and FreeBSD. This work introduces AVG for Linux/FreeBSD and its usage. Compilation and link are discussed from the source code to the distribution packages, which users can install on your computer. Also, the repository term was introduced, containing the information about their creation and usage. The part about AVG for Linux/FreeBSD testing discusses suitable automatic testing proposals for this product and implementation of the best solutions. In the practical part, the testing tool was developed. AVG for Linux/FreeBSD was tested using the test tool and implemented test set.

Klíčová slova

AVG for Linux/FreeBSD, GNU/Linux, FreeBSD, Make, balíčky, DEB, RPM, TGZ, testování, automatizace testů

Keywords

AVG for Linux/FreeBSD, GNU/Linux, FreeBSD, Make, packages, DEB, RPM, TGZ, testing, tests' automation

Citace

Jiří Suška: Framework pro sestavení a testování
bezpečnostního síťového řešení, diplomová práce, Brno, FIT VUT v Brně, 2011

Framework pro sestavení a testování bezpečnostního síťového řešení

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Vojtěcha Matesa.

.....
Jiří Suška
23. května 2011

Obsah

Kapitola 1

Úvod

Úkolem této práce je vytvořit automatický sestavovací a testovací systém, který bude sloužit k sestavení a testování bezpečnostního systému AVG for Linux/FreeBSD. Výsledkem celého procesu by měly být zvolené instalační balíčky a/nebo archivy AVG for Linux/FreeBSD a AVG for Linux/FreeBSD Free, rozdílové soubory pro aktualizace. Dále by tyto balíčky měly projít sadou testů, které prověří jejich základní funkcionalitu. Prověřeno by také mělo být, že byly balíčky správně sestaveny a nedošlo k zásadní chybě, která by způsobila nefunkčnost systému.

Úvodní 1. kapitola má za úkol seznámit čtenáře s cílem práce a seznámit ho s jejím členěním.

Po této kapitole následují 3 kapitoly teoretického charakteru. Tyto kapitoly slouží k uvedení čtenáře do problematiky sestavování a distribuce SW na platformách GNU/Linux a FreeBSD.

Kapitola 2 seznamuje čtenáře s produktem AVG for Linux/FreeBSD a stručně mu představuje funkcionalitu tohoto produktu. V samostatné podkapitole je představeno AVG Rescue CD GNU/Linux, k čemu tento produkt slouží a jeho vztah k AVG for Linux/FreeBSD.

V kapitole číslo 3 je přiblížena problematika překladu a sestavování zdrojových kódů a problematika zabalení SW systému do distribučních balíčků a archivů. První část kapitoly seznamuje čtenáře s možnostmi automatického překladu zdrojových kódů a stručně popisuje některé systémy pro automatický multiplatformní překlad, jakými jsou Make, CMake a Apache Ant. Druhá část kapitoly se zabývá možnostmi vytvoření balíčků a archivů. Je zde představeno několik druhů balíčků a archivů, které se používají pro distribuci software na platformách GNU/Linux a FreeBSD. V kapitole je rozebráno, jak jednotlivé balíčky a archivy vytvořit a jakým způsobem vytvořit úložiště (repozitář), ze kterého si mohou uživatelé balíčky stáhnout. Kapitola také stručně rozebírá nebezpečí podvržení instalátoru a seznamuje, jakým způsobem je možné uživatele před tímto nebezpečím chránit.

Poslední teoretická kapitola (4. kapitola) rozebírá možnosti testování SW a především automatizaci testování. V kapitole jsou představeny různé druhy testů podle přístupu k testované aplikaci a podle účelu. Dále je zde uveden vztah mezi manuálními a automatickými testy. Tato část se také zabývá výhodami automatických testů, jejich přínosy a situace, ve kterých je možné testy automatizovat. V závěrečné části kapitoly je čtenář odkázán na metodiku, která by mu měla pomoct se rozhodnout, zda a případně které testy automatizovat.

Kapitola 5 je rozdělena na tři části. První část se věnuje sestavování. Seznamuje čtenáře, jak fungoval původní systém sestavování a testování, jak a proč byl tento systém vylepšen. Druhá část se věnuje vývoji podpůrného testovacího nástroje a zdůvodňuje, proč bylo potřebné tento nástroj vyvinout. Třetí část kapitoly se věnuje samotnému testování produktu

AVG for Linux/FreeBSD.

Poslední kapitola, číslo 6, hodnotí dosažené výsledky, uvažuje další potenciální vývoj a možná vylepšení vytvořeného systému.

Tato práce navazuje na semestrální projekt. V rámci semestrálního projektu byl nastudován bezpečnostní systém AVG for Linux/FreeBSD, možnosti automatické kompilace a sestavení SW projektů a způsoby instalace SW na platformách Linux a FreeBSD. Tyto poznatky byly použity při návrhu úprav a vylepšení sestavovacího procesu AVG. Zmíněná vylepšení jsou popsána v kapitole 5.1. Dále byla v semestrálním projektu navržena množina testů, která byla přehodnocena a využita v kapitole 5.3.

Kapitola 2

AVG for Linux/FreeBSD

Tato kapitola má za cíl seznámit čtenáře s produktem AVG for Linux/FreeBSD (dále už jen AVG), s jeho komponentami a s produktem AVG Rescue CD GNU/Linux.

2.1 Seznámení s AVG for Linux/FreeBSD

AVG je antivirový a bezpečnostní produkt, určený primárně pro souborové a poštovní servery. Cílem tohoto produktu je zajistit pokud možno komplexní ochranu počítače (resp. počítačů propojených v počítačové síti) před hrozbami přicházejícími z Internetu a skrze počítačovou síť. AVG se snaží zabránit šíření hrozeb po síti ať už prostřednictvím e-mailu nebo infikovaných souborů.

E-maily jsou ověřovány na přítomnost virů v přílohách a je kontrolováno, zda se nejedná o spam, případně o phishingový e-mail[1].

Souborový server je chráněn pomocí AVG On-Access Daemona[2], který při požadavku o přístup k souboru rozhodne na základě antivirové kontroly, zda povolí nebo zamítne přístup.

Produkt je určen pro operační systémy Unixového typu, konkrétně pro systémy GNU/Linux a FreeBSD. V současné době jej výrobce distribuuje ve formě instalačních balíčků (resp. instalačních archivů) DEB, RPM, TGZ a SH pro GNU/Linux. Pro systém FreeBSD jsou určené instalační archivy TGZ. AVG je poskytováno ve dvou variantách AVG for Linux/FreeBSD (někdy označováno jako Full) a AVG for Linux/FreeBSD Free. Verze AVG for Linux/FreeBSD Free je možno používat bezplatně bez omezení detekčních schopností. Na rozdíl od Free, komerční varianta má komponentu antispam a je k ní poskytována technická podpora. Další rozdíl je možné najít v aktualizacích produktu. Varianta Free umožňuje provést pouze jednu aktualizaci denně [3](ráno) [16].

Produkt je poskytován v jediném jazyce. Tímto jazykem je angličtina. Dokumentace je dodávána ve dvou podobách. Jednou podobou jsou manuálové stránky, a druhou uživatelská příručka. Manuálové stránky jsou součástí instalačních balíčků a popisují jednotlivé spustitelné komponenty. Informace, jak nainstalovat a používat AVG, je dodávána v uživatelské příručce, kterou si uživatel může stáhnout z webových stránek.

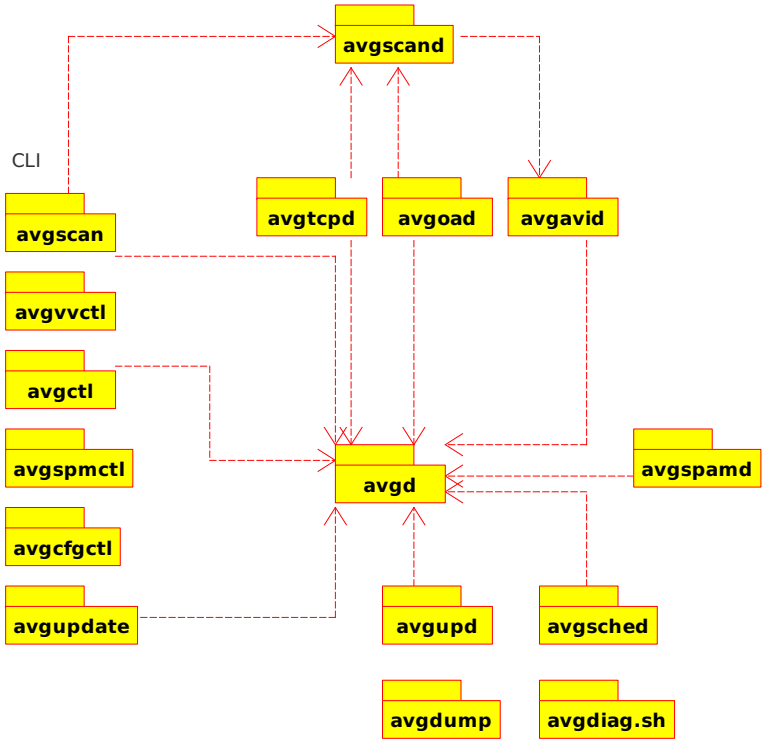
[1] phishing označuje e-maily, které se snaží z důvěřivců vylákat informace (přístupové údaje, čísla kreditních karet, atp.) tím, že se pokoušejí vypadat podobně jako e-maily známých organizací, např. bank

[2] daemon je program, který je spuštěn dlouhodobě a neinteraguje s uživatelem přes uživatelské rozhraní

[3] aktualizace se vydávají běžně ráno a večer, při výjimečných případech i vícekrát denně

2.1.1 Komponenty

AVG je možné rozdělit na několik vzájemně propojených komponent. Na obrázku 2.1 jsou znázorněny vztahy mezi těmito komponentami (diagram ukazuje, které komponenty potřebují ke své funkčnosti jiné komponenty) [7].



Obrázek 2.1: Závislosti komponent AVG – stav AVG 8.5

- avgavid – AVG AVI loader daemon. Slouží k načítání virové databáze do sdílené paměti. Toto se provádí, protože virová databáze je velký binární soubor (verze 3342 zabírá přibližně 98MB). Pokud by tato databáze sdílená nebyla, musel by si ji každý proces načíst z disku, což při této velikosti trvá dlouho a dochází ke zbytečnému zaplnění paměti RAM.
- avgcfgctl – AVG command line configuration controller slouží pro práci s konfigurací AVG. Kromě změny konfiguračních položek umožňuje export a import celé konfigurace.
- avgctl – AVG command line controller zajišťuje rozhraní příkazové řádky (CLI - command line interface) komponenty avgd. Dále slouží pro registraci licenčního čísla, výpis verzí a aktuálního stavu jednotlivých komponent, výpis verze virové databáze a ovládání některých dalších komponent (např. avgsched).
- avgd – AVG Daemon je hlavní daemon. Tento daemon se stará o spouštění, zastavování a restartování jednotlivých komponent. V případě, že některá z komponent selže, avgd ji opětovně spustí.
- avgdump – AVG crash dumper je užitečný v případech, kdy dojde k selhání některé komponenty. Do speciálních souborů vypíše stav zásobníku jednotlivých vláken hava-

rovaného procesu, konfiguraci jednotlivých komponent, dále zjistí, jaké jsou načtené knihovny, jejich verze a volitelně vytvoří core dump[4]. Tyto informace potom uživatel může poslat na oddělení technické podpory, které se pomocí nich pokusí odhalit příčinu problému. Po odhalení a odstranění problému se oprava promítne do další verze AVG.

- avgoad – AVG On-Access Daemon provádí virovou kontrolu souborů, ke kterým je přistupováno. Jedná se o program, který spolupracuje s moduly jádra operačního systému, pro odchycení požadavků přístupu k souboru. K tomuto účelu se používají moduly souborových systémů např. Dazuko [31] a RedirFS [36]. Tyto moduly přidávají novou vrstvu mezi virtual file system (VFS) a ovladače souborového systému. Zmíněné řešení umožňuje modifikovat volání ve VFS, takže pokud je soubor nakažený, dojde k odmítnutí přístupu[5].

- avgscan – AVG command line scanner je CLI komponenty avgscand. Slouží k antivirovému skenování souborů, adresářů (on-demand scan) a interaktivnímu výpisu výsledku skenování na příkazový řádek[6].

- avgscand – AVG scan daemon vyřizuje požadavky na antivirové skeny souborů dat generované anebo zasílané komponentami avgscan, avgoad a avgtcpd.

- avgsched – AVG Scheduler Daemon slouží pro spouštění naplánovaných událostí jako je aktualizace souborů, virové a nebo antispamové databáze.

- avgspamd – AVG Anti-Spam Daemon je obdobou scand při ochraně proti spamu a phishingovým e-mailům.

- avgspmctl – slouží k učení rozpoznávacího systému pro odhalování spamu. Při učení se používá předkládání spamových (spam) a nespamových (ham) vzorků e-mailů. Součástí AVG je od verze 2011.

- avgtcpd – Mail filter for virus/spam/phishing scanning slouží ke skenování e-mailů pro ochranu před viry, spamem a phishingovými emaily. Avgtcpd podporuje spolupráci s e-mailovými servery PostFix, QMail, SendMail a Exim.

- avgupd – AVG Update Daemon je komponenta, která provádí aktualizaci AVG (aktualizace souborů a virové a nebo antispamové databáze). Aktualizace je možné provést z adresáře nebo z Internetu. Aktualizace jsou vydávány s různými prioritami:
  - kritická aktualizace,
  - virová aktualizace,
  - doporučená aktualizace,
  - programová aktualizace (výchozí),
  - nepovinná aktualizace.

[4] výpis obsahu paměti spuštěného procesu v daném čase

[5] systémové volání vrátí chybu „přístup zamítnut“ (Permission Denied)

[6] samotný sken se provádí komponentou avgscand, CLI ve skutečnosti pouze generuje požadavky a zabezpečuje komunikaci s uživatelem

Aktualizace jsou vydávány jako rozdílové soubory mezi různými verzemi produktu. Mimo rozdílových aktualizací je vydávána také tzv. plná (full) aktualizace, která obsahuje kompletní zkomprimované soubory dané komponenty. Avgupd si stáhne průvodní soubory o možnostech aktualizací a vyhodnotí, zda je výhodnější (zda zabírá méně místa) stáhnout jednu nebo více rozdílových aktualizací, nebo zda stáhne plnou aktualizaci dané komponenty.

- avgupdate – AVG command line update je CLI k avgupd. Umožňuje uživatelsky vyžádat update AVG.
- avgvvctl – AVG Virus Vault (VV) controler je program provádějící operace nad AVG VV. VV je adresářová struktura, která slouží pro uložení napadených souborů do karantény. Do VV se kromě samotného souboru ukládají i informace, které jsou nutné k obnovení souboru (vlastník, skupina, ...). Tato metadata jsou ukládána odděleně od samotných souborů. Napadené soubory nemohou být prostě smazány, uživatel by přišel o informace, které soubor obsahuje a nebylo by možné zjistit, kdy byl soubor pravděpodobně napaden. Aby nedocházelo k šíření infekce, nejsou soubory uloženy ve své původní podobě, ale jsou překódované. Avgvvctl se používá k výpisu obsahu VV. Soubory a adresáře uložené ve VV mohou být pomocí avgvvctl odstraněny nebo obnoveny. VV je optimalizován na rychlost. Operace vkládání, obnovování a mazání mají konstantní složitost. Operace výpisu obsahu VV má lineární složitost. Program komunikuje s uživatelem pomocí CLI.

2.2 AVG Rescue CD GNU/Linux

Na základě AVG for Linux je postaveno i AVG Rescue CD GNU/Linux (ARL). Jedná se o live CD[7] vytvořené pomocí BuildRoot [30] s nainstalovanými ovladači různých zařízení, nainstalovaným upraveným AVG for Linux (neobsahuje některé komponenty jako je antispam, aj.) a podpůrnými aplikacemi pro správu systému jako je program pro administraci Virus vaultu, který používá produkt AVG 2011 for Windows[8]. Ovládané je pomocí textového uživatelského rozhraní (TUI[9]) a jeho hlavní funkcí je umožnit uživateli opravit havarovaný operační systém (např. umožní uživateli provést antivirový test, opravit poškozený souborový systém, přesunout soubory chybně vložené do karantény zpět na původní místo, ...). Produkt se sice jmenuje Rescue CD, ale mimo CD je možné jako nosič použít i USB flash disk. ARL je zcela soběstačné, takže nevyžaduje žádnou podporu ze strany operačního systému ani jiného software, který je na počítači nainstalován. ARL vypálené na CD nebo DVD není perzistentní, takže všechny změny, které jsou na něm provedeny (např. stažení nové virové databáze), jsou po restartu ztraceny. Flash disk s nainstalovaným ARL perzistentní je. Neperzistence CD/DVD se týká pouze ARL. Změny, které jsou provedeny na pevném disku jsou trvalé. Mimo TUI obsahuje ARL i plnohodnotný příkazový řádek, který především zkušeným uživatelům umožňuje provádět pokročilá nastavení, změny a další kroky vedoucí k záchraně jejich operačního systému. Protože ARL nepoužívá operační systém uložený na pevném disku počítače, je snadnější provést sken, identifikovat a odstranit viry a rootkity (tyto škodlivé programy nejsou spuštěné, takže nemohou provádět opatření, pomocí kterých by se skryly nebo přesunuly).

[7] CD s operačním systémem, který je možné spustit bez nutnosti instalace na pevný disk počítače

[8] tento VV nemá shodnou strukturu jako VV AVG for Linux a nástroje k jeho ovládání nejsou součástí běžného AVG for Linux

[9] text user interface

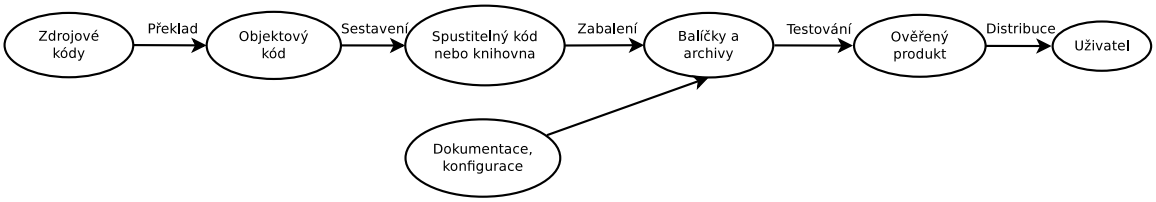
ARL je distribuováno zdarma formou obrazů CD/USB flash disku a zdrojových souborů CD, včetně Makefile pro GNU Make (zdrojovými soubory zde není myšlen pouze zdrojový kód, protože AVG for Linux je vkládáno ve formě předkompilovaných souborů, ale ostatní komponenty, jako je TUI, jsou dodávány včetně zdrojových kódů), o kterém je více napsáno v částech 3.1.1 a 3.1.4. Všechny komponenty ARL kromě AVG jsou distribuovány pod svobodnými licencemi, jako je např. licence GNU/General public license (GNU/GPL) [15].

Kapitola 3

Překlad, sestavování a zabalení SW

Kapitola uvede čtenáře do problematiky sestavování SW ze zdrojových kódů. Dále se věnuje popisu instalačních balíčků a archivů, detailně popisuje postup jejich tvorby, seznamuje s pojmem repozitář a s jeho vztahem k balíčkům. Informuje o možnostech podvržení instalátorů a naznačuje, jak se tomuto nebezpečí vyhnout pomocí podpisu GNU Privacy Guard [34] (GnuPG[1]).

Cesta software k uživateli, od zdrojových kódů k výslednému produktu, který si uživatel na svůj počítač nainstaluje, prochází několika fázemi. Cestu můžeme rozdělit na překlad, sestavení a zabalení. Na obrázku 3.1 je znázorněno, jak na sebe tyto fáze navazují.



Obrázek 3.1: Cesta programu k uživateli

3.1 Překlad a sestavování

Překlad je činnost, při které překladač (kompilátor) převádí zdrojový kód napsaný programátory v programovacím jazyce, na instrukce strojového kódu, které je schopen vykonat mikroprocesor počítače. Přeložený kód je na platformně závislý, což znamená, že kód přeložený pro jednu architekturu mikroprocesoru nebude fungovat na mikroprocesoru s jinou architekturou[2] a kód přeložený pro jeden operační systém nemusí být spustitelný na jiném OS[3].

Výsledkem překladu může být spustitelný soubor, častěji je však výsledkem objektový kód. Z několika souborů, které obsahují objektový kód, vytváří linker (sestavovací program)

[1] GnuPG je svobodná implementace standardu OpenPGP, který je standardizován v RFC4880. Navíc je běžnou součástí většiny distribucí GNU/Linuxu i OS FreeBSD

[2] toto je způsobeno rozdílností instrukčních sad různých architektur mikroprocesorů

[3] mohou používat jiný formát spustitelného souboru (např. GNU/Linux, FreeBSD a další Unixové OS používají formát Executable and Linkable Format - ELF, na rozdíl od OS MS Windows, které používají formát Portable Executable - PE) a/nebo jiný formát jaderného ABI (application binary interface)

sestavením (linkováním[4]) spustitelný soubor nebo knihovnu. Toto je výhodné v případech, kdy jeden modul objektového kódu sdílí více výsledných spustitelných souborů. Linkování prováděné při překladu se nazývá statické linkování. Mimo statického linkování existuje ještě dynamické linkování. Dynamické linkování neprobíhá při překladu, ale při spuštění programu. Používá se pro přilinkování dynamických knihoven[5] k spouštěnému programu. Výhodou dynamických knihoven je, že mohou být sdíleny více programy (nachází se na disku v jedné kopii). Navíc mohou být aktualizovány bez aktualizace programu. Z této vlastnosti vyplývá i jejich nevýhoda. Díky aktualizaci se rozšiřuje množství konfigurací cílového systému, na kterém uživatelé programy používají, takže se mohou projevit i chyby, které se při testování na jiné konfiguraci neprojevily.

Překládání programu je časově náročná procedura. Pokud bychom překládali při každé změně celý zdrojový kód, docházelo by u velkých projektů k neefektivnímu využití času programátora. Za předpokladu, že bychom chtěli „ručně“ překládat pouze některé části, které jsme změnili, mohlo by dojít k chybě lidského faktoru (člověk by zapomněl, že v tomto objektovém souboru také používá tuto knihovnu, ...). Tento problém řeší programy pro automatický překlad jako je Make, Apache Ant, CMake, GNU Automake, ...

3.1.1 Make

Make je program pro automatický překlad a sestavování programů. Byl vytvořen v roce 1977 v Bellových laboratořích Stuartem Feldmanem. Jde o jednoduchý systém, který na základě pravidel napsaných a uložených v souboru pojmenovaném (obvykle) `Makefile` přeloží potřebnou část zdrojových souborů a slinkuje pozměněné objektové soubory do daného cíle (programu nebo knihovny). Make je souborově orientovaný program. To znamená, že pravidla napsaná v `Makefile`, popisují jak vytvořit soubory (cíle), které závisí na jiných souborech (závislosti). Pro příklad syntaxe viz 3.1.

```
vysledky: zavislosti
        prikazy
```

Příklad 3.1: Příklad syntaxe Make

Pravidlo uvedené v tomto příkladu popisuje, jak soubory uvedené v seznamu *výsledky* vytvořit pomocí shellového skriptu *příkazy*. Soubory v seznamu *výsledky* závisí na souborech uvedených v seznamu *závislosti*. Skupina několika takových pravidel potom vytváří strom závislostí. Celý systém funguje na principu časových razítek souborů. Pokud si vyžádáme sestavení souboru Make:

1. sestaví strom závislostí opakovaným procházením `Makefile`;
2. prověří prvky ve stromu (soubory) podle časových razítek, zda nejsou starší než jeho následovníci:
   - pokud některých z nich je starší, tak danou část stromu přeloží a znovu sestaví;
   - pokud některý z prvků ve stromu chybí na disku úplně, musí existovat pravidlo, jak jej vytvořit. Toto pravidlo se aplikuje a tím je vytvořen nový soubor, tedy

[4] sestavování objektových souborů

[5] knihovna, která se při překladu nelinkuje přímo k programu, vkládají se pouze symboly, které obsahuje. Při spuštění programu je (na OS GNU/Linux) nahrána do paměti knihovna `ld.so`, která se postará o nahrání ostatních sdílených knihoven do paměti.

soubor s časem úpravy novějším než soubory původní, a proto musí být jeho předchůdci také znovusestaveni.

Existuje několik implementací Make např. BSD Make, GNU Make [17], ... Make je součástí standardu POSIX [22].

Výhodou Make je, že se vždy automaticky sestavují jen ty dané části stromu, které jsou potřeba znovusestavit (protože se změnily soubory, na kterých jsou závislé). Uživatel pouze zadá cíle automatické kompilace a sestavení. Za další výhodu můžeme považovat, že Make používá příkazy shellu, takže pomocí jednoho `Makefile` je možné popsat, jak sestavit program ze zdrojových kódů napsaných v jazyce C, dokumentaci pomocí Doxygen[6], atd.

Nevýhodou Make je, že si negeneruje pravidla automatickým způsobem[7], takže pokud programátor zapomene zapsat závislost do `Makefile`, překlad nebude fungovat správně[8]. Některé překladače, např. GNU Compiler Collection (GCC) [33], umožňují nastavit generování stromu závislostí a uložit strom ve formě souborů. Díky tomu je možné se v Make pravidlech odkazovat na tyto soubory a mít zajištěno správné nastavení závislostí. Make také neřeší odlišnosti různých systémů, na kterých by mohl být program kompilován, proto je nutné zohlednit tyto rozdílnosti (různé parametry překladačů na různých architekturách procesorů, různé cesty k systémovým knihovnám, ...). Pokud chceme vytvořit jeden univerzální Makefile pro více OS (např. GNU/Linux a FreeBSD), musíme buď v `Makefile` napsat podmíněné části pro nastavení parametrů, nebo použít některý z generátorů těchto parametrů např. GNU Autoconf [28] a GNU Automake [29] popsané níže v části 3.1.4.

3.1.2 Apache Ant

Dalším nástrojem pro automatickou kompilaci a sestavení je Apache Ant (Another Neat Tool). Je to nástroj napsaný v jazyce Java. Jako konfigurační soubor projektu používá `build.xml`. Tento soubor je ve formátu XML. Jedná se o nástroj, který převažuje u projektů napsaných v jazyce Java. Některá IDE (např. Netbeans) jej dokonce používají jako hlavní formát pro správu projektů. Apache Ant byl původně napsán Jamesem Duncanen Davidsonem pro sestavení servletového kontejneru Apache Tomcat. Samostatná verze 1.1 byla vydána 19. července 2000 [20].

Apache Ant má oproti Make několik výhod. První je, že pro sestavení cílů nepoužívá platformně závislé příkazy shellu, ale používá vnitřní funkce. Dále řeší problém oddělovače v adresářové cestě, která se může v závislosti na platformě lišit (např. v případě Unixových systémů se používá symbol '/' a v případě systémů z rodiny MS Windows se používá symbol '\'). Apache Ant umožní uživateli zvolit preferovaný oddělovač, ale pokud jej na dané platformě nelze použít, vybere jiný. Tyto problémy jsou sice řešitelné i v Make, ale je nutné problém řešit při psaní Makefile podmínkami, nastavením z příkazové řádky, ...

3.1.3 CMake

Dalším programem pro automatizovaný multiplatformní překlad a sestavení je CMake (Cross Platform Make). Jedná se o open-source systém [21]. CMake byl vyvinut v roce 1999 a implementován v roce 2000.

[6] program pro automatické generování dokumentace ze zdrojových kódů [32]

[7] toto není přesné, Make umožňuje generovat pravidla automaticky, ovšem ne ve smyslu procházení zdrojových souborů

[8] závislosti nebudou fungovat správně, takže nelze zaručit, že skutečně bude znovusestaveno vše co by na daném souboru mělo záviset

Stejně jako nástroje ze skupiny GNU Autotools popsané v části 3.1.4 i CMake je systém pro automatizování překladu multiplatformního kódu. Výhodou CMake je, že umožňuje generovat mimo Makefile i další výstupy, např. projektové soubory Apple Xcode a Microsoft Visual Studio. Program CMake je napsaný v C++ s minimálními závislostmi na externích knihovnách.

Pro nastavení závislostí a požadavků CMake používá textový soubor `CMakeLists.txt`, kterým je řízen překlad na požadovaný typ projektového souboru.

3.1.4 GNU Automake, GNU Autoconf a GNU Make

Aby bylo možné přeložit program na různých operačních systémech, je nutné dodržovat jisté zásady a používat vhodné knihovny. Mimo to je nutné zajistit i překlad na cílovém systému. Na Unixových operačních systémech se pro automatizovaný překlad používá Make. Ruční psaní `Makefile` programu Make pro každý systém, na kterém má být program kompilován, není pohodlné. Na různých systémech se liší např. umístění a verze knihoven, parametry překladače, překladač programovacího jazyka, cesta k hlavičkovým souborům, definice datových typů, ... Před samotným překladem je proto nutné nastavit všechny zmíněné proměnné a mnoho dalších parametrů.

K ověření přítomnosti a nastavení těchto proměnných slouží nástroj GNU Autoconf. Program pro GNU Autoconf se programuje v jazyce m4 a obvykle se ukládá do souboru `configure.in`. Po překladu souboru `configure.in` vznikne soubor `configure`. Tento soubor obsahuje příkazy shellu, které testují, zda jsou přítomny vyžadované programy, knihovny, hlavičkové soubory, ...

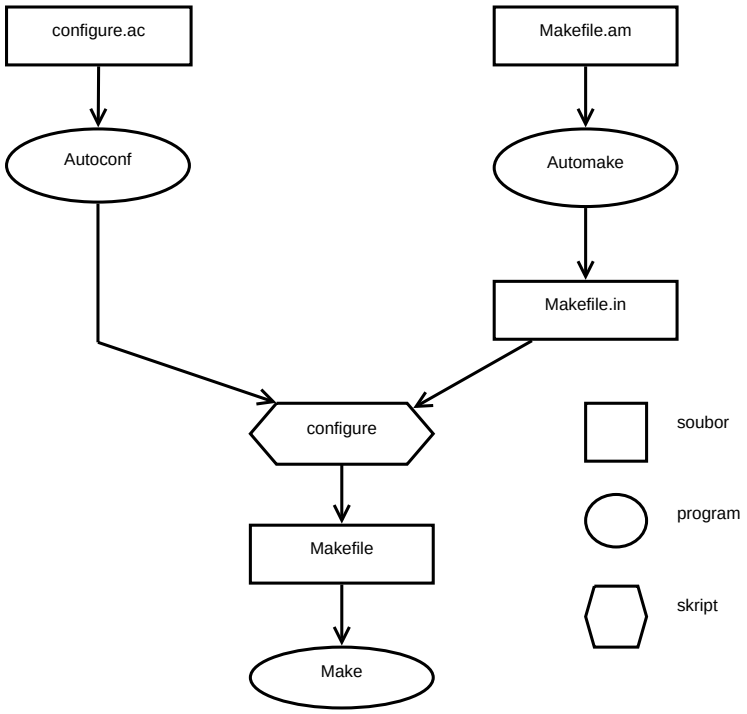
S GNU Autoconf spolupracuje program GNU Automake. Tento nástroj, z šablon uložených v souboru `Makefile.am`, vytvoří pravidla, která uloží do souboru `Makefile.in`. Pravidla jsou po spuštění skriptu `configure` doplněna o proměnné, které ověřil a nastavil GNU Autoconf. Výsledkem spuštění `configure` je soubor `Makefile` (standardní `Makefile` pro program GNU Make) [18].

GNU Make je svobodnou implementací programu Make, který je blíže popsán v části 3.1.1. K funkcionalitě Make přidává např. šablony, pomocí kterých lze generovat pravidla pro sestavení cílů. Postup při překladu programu je znázorněn na obrázku 3.2.

Funkcionalita představených programů je doplněna celou řadou dalších nástrojů pro automatizovaný překlad. Tyto nástroje jsou zařazeny do skupiny programů GNU Autotools. O těchto programech je možné zjistit více např. na webových stránkách [19].

3.2 Příprava distribučních balíčků a archivů

Potom co byl program sestaven, je možné jej zveřejnit. Zveřejňování může probíhat mnoha způsoby. Je možné zkompilovaný program se všemi jeho dodatečnými součástmi zveřejnit pomocí webového nebo ftp serveru. Jenže tímto způsobem bychom uživateli připravili horké chvilky při jeho instalaci a používání. Po stažení by musel soubor(y) uložit do vhodného adresáře, vytvořit si symbolické odkazy, přesunout konfigurační soubory a skripty na příslušné místo, atp. Toto je velice nepohodlné. Vyžadovalo by to u uživatele nemálo zkušeností, dovedností a trpělivosti. Důsledkem by pravděpodobně bylo, že by někteří uživatelé byli procedurou přípravy programu natolik odrazeni, že by program raději nepoužívali. Z těchto důvodů existuje několik cest, jak program uživateli poskytnout a získání software zpříjemnit. Standardní cesty, které jsou k dispozici na Unixových OS, je možné rozdělit na archivy a balíčky.



Obrázek 3.2: Postup kompilace pomocí GNU Automake, GNU Autoconf a GNU Make

3.2.1 Archivy

Instalační archivy jsou zkomprimovanou podobou šířeného software. Kromě samotného software mohou obsahovat skripty, které slouží k instalaci a následné odinstalaci. Vše, co je součástí archivu, si řeší správce archivu ve své režii (např. právě zmíněná instalace a odinstalace). To lze ale chápat i jako výhodu, protože správce archivu není limitován ničím předepsaným a povinným (na rozdíl od balíčků, které jsou popsány dále v části 3.2.2), takže není ničím omezován.

3.2.2 Balíčky

Dalším způsobem, jak distribuovat software na OS GNU/Linux (případně FreeBSD), je využití některého z balíčkovacích systémů. Balíčkovací systém slouží pro správu software, který je připraven ve speciálních instalačních souborech nazývaných balíčky. Tento systém obvykle umožňuje instalaci, odinstalaci a aktualizaci balíčků. Další funkcí balíčkovacího systému je možnost vytvářet závislosti mezi balíčky[9].

Závislosti slouží k nastavení pevných a volných vazeb mezi balíčky. Příkladem pevné vazby je závislost na knihovnách OS (zabalený program je totiž ke svému spuštění vyžaduje). Volná závislost se používá k závislosti na manuálových stránkách nebo jiných programech, které ke spuštění program nutně nepotřebuje, ale uživatel by měl vědět, že instalací takového balíčku získá další funkce nebo součásti, které k základnímu spuštění nejsou nutné. Příkladem využití volných závislostí může být rozdělení software na funkční bloky, díky čemuž můžeme od jádra SW, které je k základní funkcionalitě nezbytně nutné, oddělit volitelné součásti, např. plug-iny nebo dokumentaci. Díky závislostem také není

[9]resp. mezi programy v případě některých balíčkovacích systémů

nutné instalovat některé balíčky vícekrát, např. sdílené knihovny mohou používat v jedné verzi všechny programy. Tato vlastnost je velice žádoucí, protože balíček nemusí tuto knihovnu obsahovat. Postačí, když bude obsahovat pouze program a na knihovny, případně další software (např. interpret skriptovacího jazyka), se odkáže závislostí. Díky tomu OS vybavené balíčkovacím systémem umožňují uživateli velkou volnost při instalaci různého SW v různých verzích. Tato vlastnost má negativní důsledek pro technickou podporu. Díky velké variabilitě nainstalovaných spolupracujících programů a knihoven různých verzí, není vždy snadné vyvolat stejné chování, jaké se projevuje u uživatele.

Pokud nahlédneme na balíčkovací systémy z pohledu poskytovatele SW zjistíme, že existuje větší množství různých balíčkovacích systémů, a proto je nemožné použít jeden typ balíčků a zároveň pokrýt většinu potenciálních zákazníků. Dalším problémem je, že neexistuje mnoho zdrojů, které by nám naznačily, jaké druhy balíčků zvolit pro pokrytí co největšího množství uživatelů. Informace můžeme čerpat z analýzy statistik rozšíření jednotlivých distribucí GNU/Linuxu generovaných pomocí statistik návštěvnosti webových stránek, např. [12] (v tabulce 3.1), dále ze stránek zabývající se zobrazováním informací od registrovaných uživatelů [14] (tabulka 3.2) a z anket odborně zaměřených webových stránek [13] (tabulka 3.3).

Z tabulek 3.1, 3.2 a 3.3 lze vyvozovat, že nejrozšířenějšími typy balíčků jsou DEB a RPM.

Název distribuce	Počet návštěv [osob/den]	Počet návštěv [%]	Typ balíčků
Ubuntu	2182	7,24	DEB
Fedora core	1549	5,14	RPM
Mint	1510	5,01	DEB
OpenSUSE	1234	4,09	RPM
Debian	1049	3,48	DEB
PCLinuxOS	931	3,09	RPM
Mandriva	890	2,95	RPM
Sabayon	818	2,71	archivy

Tabulka 3.1: Výběr osmi distribucí GNU/Linuxu, ze statistiky návštěvnosti serveru http://distrowatch.com [12], které používá nejvíce návštěvníků

3.2.3 Rozšířenost balíčkovacích systémů

Z tabulek 3.1, 3.2 a 3.3 vyplývá, že abychom pokryli co největší množství počítačů s operačními systémy GNU/Linux a FreeBSD, nemůžeme se spolehnout na jeden typ balíčku. Problém by bylo možné řešit distribucí archivů obsahujících program, ale tím by uživatelé přišli o komfort, který jim přináší použití balíčkovacího systému. Naštěstí je tvorba jednotlivých druhů distribučních archivů/balíčků časově nenáročná a implementačně snadná. Jediným problémem, který z tvorby plyne, je nutnost ověřit správnou funkčnost všech vytvořených archivů/balíčků a porovnat binární shodnost nainstalovaného SW.

3.2.4 Nebezpečí pro uživatele

Existuje reálné nebezpečí podvržení programu a jeho součástí útočníkem. Útočník připraví instalátor (balíček nebo archiv), o kterém bude tvrdit, že obsahuje program, který uživatel

Název distribuce	Počet registrovaných [osob]	Počet registrovaných [%]	Typ balíčku
Ubuntu	31896	24,86	DEB
Ostatní	24399	19,02	-
Debian	20764	16,19	DEB
SUSE	9803	7,64	RPM
Slackware	8441	6,58	TGZ
Fedora core	8075	6,29	RPM
Gentoo	7938	6,19	Zdrojové kódy, ebuily
Red Hat	5963	4,65	RPM

Tabulka 3.2: Distribuce GNU/Linuxu na počítačích registrovaných na http://counter.li.org [14]

Název distribuce	Hlasovalo [%]	Typ balíčku
Ubuntu	44,3	DEB
Debian	19,5	DEB
Fedora core	11,5	RPM
Arch Linux	11,4	archivy
Mandriva	10,3	RPM
SUSE	9,8	RPM
Gentoo	9,6	Zdrojové kódy, ebuily
Linux Mint	4,5	DEB

Tabulka 3.3: Anketa AbcLinuxu o nejoblíbenější distribuci 2010 [13]

chce[10]. Uživatel si takový instalátor stáhne, nainstaluje a následně jej spustí. Tím byl spuštěn útočníkův kód, který může na počítači napáchat škody, odcizit data, nebo provádět jinou škodlivou činnost. Případů, kdy došlo k podvržení instalátoru uživatelům existuje celá řada. Jedním z takových případů bylo např. vietnamské jazykové rozšíření pro prohlížeč Mozilla Firefox (Vietnamese Language Pack 2.0), které obsahovalo virus. K detekci, nahlášení a odstranění viru došlo 2 měsíce po vystavení rozšíření [39].

Proti této hrozbě je možné uživatele chránit podepisováním instalátorů. Podepisovat můžeme např. pomocí programu GnuPG. Pomocí podpisu zajistíme, že si uživatel bude mít možnost ověřit, zda jsme instalátor skutečně vydali my, nebo se mu někdo pokouší podvrhnout svůj instalátor.

3.2.5 Archivy TGZ

Archivy jsou komprimované soubory, které slouží pro distribuci binárních programů a zdrojových kódů. Jedním z druhů archivů je archiv TGZ. Jedná se o komprimovaný soubor s příponou `.tar.gz` (tarball, používá se i příloha `.tgz`).

Výhodou TGZ je, že pro svou jednoduchost umožňuje použití na libovolné distribuci GNU/Linuxu při zachování binární kompatibility. Archiv by měl obsahovat skripty pro in-

[10]toto se netýká pouze instalátorů, ale obecně libovolného souboru, který je ochoten uživatel stáhnout a později otevřít

stalaci a odinstalaci daného programu. Nevýhodou tohoto způsobu distribuce je, že neexistuje centralizovaná správa nainstalovaných programů. Navíc vyžaduje interakci při instalaci (po rozbalení je nutné spustit instalační skript). Dále jej nelze odinstalovat, ani aktualizovat, bez příslušného skriptu. Další nevýhodou, plynoucí z povahy archivů, je nemožnost automaticky specifikovat závislosti na jiných programech a knihovnách. Tento problém má dvě možná řešení:

1. Vyjmenovat prerekvizity obsahu archivu, které je nutné splnit, aby obsah správně fungoval, tzn. problém přesunout na uživatele;

2. Přiložit všechny nutné součásti přímo do archivu a tím zvětšit celkovou velikost archivu.

Za další nevýhodu je možné považovat, že každý archiv (resp. instalační a jiné pomocné skripty) se může chovat jinak a instalační akce i kontroly jsou plně v režii poskytovatele archivu a toho, kdo si jej chce nainstalovat. Toto může vést až k podsunutí škodlivého kódu do balíčku.

Pro vytvoření archivu stačí mít připraven program, všechny jeho součásti a skripty pro instalaci/odinstalaci (v případě, že je chceme uživateli poskytnout) v adresáři. Tento připravený adresář stačí zabalit archivovacím programem, jako v příkladu 3.2. Příklad instalace je

```
tar -Rczf balíček.tar.gz PřipravenýAdresář
```

Příklad 3.2: Vytvoření archivu TGZ

uveden v příkladu 3.3. Pokud chceme uživateli umožnit program odinstalovat, je vhodné do

```
tar -xzf balíček.tar.gz
./PřipravenýAdresář/install.sh
```

Příklad 3.3: Instalace archivu TGZ

archivu přibalit odinstalační skript, který v případě spuštění korektně odinstaluje program z počítače uživatele.

Podepisování

Jednou z možností jak ochránit uživatele před podvržením archivu je podepisování. Používá se pro kontrolu autentičnosti archivu. Podpis zajistí, že archiv byl vydán skutečně vydavatelem (kontrolou klíče) a že nebyl upravován. Jelikož archiv TGZ sám o sobě žádnou podporu ověřování podpisu nenabízí, je nutné řešit podepisování odděleně např. pomocí GnuPG.

Podpis můžeme vytvořit pomocí příkazu v příkladu 3.4. Výstupem programu je soubor,

```
gpg -u idUživatele --armor --sign souborKPodpisu
```

Příklad 3.4: Vytvoření podpisu

který obsahuje podpis odpovídajícího archivu. Tento soubor je potom možné distribuovat spolu s archivem.

Uživatel může podpis ověřit pomocí príkladu 3.5. Tímto způsobem uživatel ověřil, že podpis odpovídá balíčku, a že podpis vydala osoba, které důvěřuje.

```
gpg --verify podpis podepsanýSoubor
```

Příklad 3.5: Ověření podpisu souboru

3.2.6 Archiv SH

Jedná se o samoinstalační soubor, který pro větší uživatelský komfort připravil Stephane Peter. Vytváří se pomocí skriptu `makeself.sh` [26]. Tento soubor je z pohledu uživatele obdobou instalačních souborů, tak jak jsou používány v MS Windows. Jde v podstatě o jednoduchý archiv. Má obdobné vlastnosti jako archiv TGZ (popsaný v části 3.2.5) s tím rozdílem, že jej není třeba při instalaci rozbalovat (resp. archiv se rozbalí sám).

Instalační soubor je možné vytvořit spuštěním `makeself.sh` s patřičnými parametry dle příkladu 3.6.

```
./makeself.sh Program ,,NázevProgramu‘‘ InstalacniSkript
```

Příklad 3.6: Vytvoření archivu SH

Kde:

- Program je cesta k adresářové struktuře, která má být zabalena.
- InstalacniSkript je cesta ke skriptu, který má být spuštěn při instalaci.

Instalace probíhá spuštěním tohoto souboru. Soubor se po spuštění rozbalí a spustí instalační skript specifikovaný při vytváření archivu (instačníSkript viz výše).

Odinstalace je prováděna stejně jako v případě archivu TGZ.

Podepisování

Podepisování probíhá stejně jako v případě archivu TGZ.

3.2.7 Balíčky DEB

Balíčky DEB jsou typem balíčku, který byl vyvinut pro distribuci GNU/Linuxu Debian. V dnešní době je tento typ balíčků používán nejen v Debianu, ale i na jeho derivátech[11] např. Ubuntu (časovou osu distribucí GNU/Linuxu můžete nalézt na [23]), ale i na mnoha dalších distribucích GNU/Linuxu a dalších operačních systémech, na které byl portován. Tento typ balíčku je AR archiv[12], který obsahuje program k instalaci, kontrolní skripty (skripty pro kontrolu chování při manipulaci s balíčkem – instalace, odinstalace, aktualizace, ...) a konfigurační soubory balíčku.

Existují dva základní druhy balíčků DEB, binární a zdrojové. Binární balíčky slouží k instalaci obsahu pro uživatele bez dalších úprav. Zdrojové balíčky slouží k distribuci zdrojových kódů, včetně úprav pro cílovou distribuci, a zabalený obsah je nutné před spuštěním přeložit, viz část 3.1. Tyto druhy balíčků se od sebe liší formátem.

Binární balíčky obsahují soubory ve spustitelné podobě a další soubory, jako jsou konfigurace manuálové stránky, apod. Dále obsahují soubory, které slouží pro popis balíčku.

[11] distribuce, které svůj základ čerpají z jiné distribuce

[12] GNU archiver - je jednoduchý Unixový nástroj, který umožňuje ze skupiny souborů vytvořit jeden soubor (archiv) a tento soubor převést zpět na původní skupinu souborů

Tyto balíčky je možné nainstalovat pomocí programu *dpkg*, případně některé z jeho nástaveb. Instalace balíčku je provedena rozbalením archivu `data.tar.gz` do kořenového adresáře operačního systému a spuštění příslušných skriptů před, resp. po, instalaci. Obsah balíčku [10]:

- `debian-binary` – obsahuje verzi formátu DEB (aktuálně 2.0);
- `control.tar.gz` – archiv obsahuje skripty a konfigurační soubory balíčkovacího systému:
  - `md5sums` – soubor obsahuje md5 hashe[13] všech souborů z archivu `data.tar.gz`;
  - `control` – obsahuje informace o balíčku:
    * Package – název balíčku (povinný);
    * Architecture – cílová architektura, pro kterou byl program připraven (povinný);
    * Section – určuje zařazení programu do sekce;
    * Priority – tento parametr naznačuje, jak důležité je pro uživatele si tuto aplikaci nainstalovat;
    * Installed-Size – velikost po nainstalování;
    * Maintainer – správce balíčku (povinný);
    * Depends – závislosti na dalších balíčcích (balíčky uvedené v tomto seznamu musí být nainstalovány, aby mohl být nainstalován tento balíček) včetně verzí těchto balíčků;
    * Recommends – tímto parametrem správce balíčku říká, že uživatelé, kteří balíček používají, obvykle chtějí instalovat i balíčky uvedené v tomto seznamu;
    * Suggests – udává, že balíčky uvedené v tomto seznamu rozšiřují funkcionalitu instalovaného balíčku;
    * Conflicts – tento balíček nebude fungovat, pokud je nainstalován některý z balíčků uvedených v tomto seznamu;
    * Replaces – tento balíček odstraňuje nebo nahrazuje soubory balíčků uvedených v tomto seznamu;
    * Breaks – balíčky uvedené v tomto seznamu nemohou být nainstalovány, pokud má být nainstalován tento balíček;
    * Provides – definuje balíčky, které jsou obsaženy v tomto balíčku (všechny jejich soubory a funkcionalita jsou součástí tohoto balíčku);
    * Homepage – lokace mainstream verze zabaleného SW (prvotní vystavení);
    * Source – určuje, který zdrojový balíček odpovídá tomuto binárnímu balíčku;
    * Version – verze programu (povinný);
    * Essential – tento parametr určuje, zda je balíček nutný pro chod systému. Pokud je nastaveno na 1, tak balíček nebude možné ze systému odstranit;
    * Description – popis balíčku (povinný);
  - `preinstall`/`preuninstall` – skript, který má být spuštěn před instalací (resp. odinstalací);

[13] číslo, vytvořené pomocí hashovací funkce (funkce, která na základě vstupu libovolné délky vrátí výsledek o pevně dané velikosti)

  - `postinstall/postuninstall` – skript, který má být spuštěn po instalaci (resp. odinstalaci);
  - `copyright` – licenční ujednání zabaleného programu a případně licenční ujednání všech souborů distribuovaných balíčkem, pro které platí jiná licence, než je licence zabaleného programu;
  - `conffiles` – seznam všech konfiguračních souborů;
  - `changelog` – obsahuje seznam změn obsahu balíčku;

- `data.tar.gz` [14] - archiv obsahuje instalovaný program.

Balíčky se zdrojovými kódy obsahují:

- soubor `.dst` – popisuje balíček;
- `.orig.tar.gz` – obsahuje komprimované zdrojové soubory v nezměněné podobě;
- `.diff.gz` – obsahuje komprimované změny zdrojových souboru.

Zdrojový balíček je možno nainstalovat pomocí programu *dpkg-source*.

Balíčky DEB je možné konvertovat pomocí programu *alien*[15] na jiné typy balíčků (RPM, TGZ, ...).

Tvorba balíčku

Protože je balíček složen pouze z jednoduchých archivů, není nutné, aby byl použit nějaký jiný software, než programy pro tvorbu archivů. Pokud ale nechceme vytvářet balíček ručně, existuje několik programů, které jsou určeny přímo pro tvorbu balíčku. Jedním z těchto programů je *dpkg-deb*. Příklad spuštění *dpkg-deb* dle příkladu 3.7.

```
dpkg-deb --build Adresář Název.deb
```

Příklad 3.7: Vytvoření balíčku DEB

Parametr Adresář je cesta k adresářové struktuře připravené k zabalení. Struktura obsahuje adresáře s programem připraveným k zabalení v adresářové struktuře odrážející absolutní cestu tohoto programu vůči kořenovému adresáři a adresář `DEBIAN` (obsahuje soubory, které budou zabaleny do archivu `control.tar.gz`). Název.deb je název výsledného balíčku, který by měl korespondovat s pravidly pro pojmenovávání balíčku:

```
NázevProgramu_VerzeProgramu-RevizeDistribuce_Architektura.deb
```

Pro více podrobností viz [10].

Podepisování balíčku

Podpis celého balíčku je podpisem obsahu balíčku, tedy tří archivů. Podpis můžeme provést ručně:

1. balíček rozbalíme,

[14] případně některý z jiných druhů archivů tar, tar.bz2, tar.lzma nebo tar.xz
[15] program pro konverzi mezi různými typy balíčků, více na [6]

2. vytvoříme jeden soubor spojením obsahu balíčku,

3. podepíšeme soubor vytvořený v kroku 2,

4. balíček opět zabalíme včetně podpisu.

Dalším způsobem, jak podepsat balíček, je použít nějaký program pro tento účel napsaný, např. *debsigs*. Balíček podepíšeme pomocí příkazu uvedeného příkladu 3.8,

```
debsigs --sign=TypPodpisu balík.deb
```

Příklad 3.8: Podpis balíčku DEB

kde TypPodpisu je jeden z trojice:

- origin – oficiální podpis organizace, která balíček vydala;
- maint – podpis správce balíčku;
- archive – podpis automaticky obnovovaný pro zajištění posledního podpisu organizace.

K podpisu bude použit výchozí GnuPG klíč [8].

Podpis je možné ověřit pomocí *debsig-verify*[16]. Výchozí nastavení některých distribucí neověřuje klíče automaticky při instalaci. Pro automatické ověřování je nutné změnit nastavení v souboru `/etc/dpkg/dpkg.cfg`. Povolením automatického ověřování zároveň znemožníme instalaci nepodepsaných balíčků, což může být problém. Tento problém je způsoben tím, že podepisování balíčků není v současné době používáno a místo něj se používá téměř výhradně podepisování repozitářů.

Repozitáře

Aby bylo instalování balíčků co nejjednodušší, jsou balíčky ukládány do úložišť – repozitářů. Jedná se o webové nebo ftp úložiště s předepsanou (doporučenou) adresářovou strukturou a komprimovanými soubory s informacemi o uložených balíčcích a případně dalšími soubory s informacemi, např. podpis repozitáře.

Založení repozitáře můžeme udělat opět ručně, nebo s použitím některého z nástrojů, např. *reprepro*. Popis tvorby repozitáře je podrobně rozepsán v [4].

Stejně jako v případě balíčků je možné, aby byl i repozitář chráněn podpisem [11]. V dnešní době se tento postup používá téměř výhradně. V případě, že balíček má být zařazen do oficiálního repozitáře a byl podepsán, je podpis z balíčku odstraněn, protože by se jednalo o duplicitní ochranu na dvou úrovních shodným způsobem.

Aby nebylo nutné zajišťovat závislosti na balíčcích ručně (balíček X závisí na balíčku Y, takže před instalací X musíme stáhnout Y a nainstalovat jej), existují nástroje, které prohledávají repozitáře. Na základě hledání potom navrhují stažení a instalaci balíčků, na kterých instalovaný balíček závisí. Základním nástrojem pro práci s repozitáři je v případě balíčků DEB annotation processing tool (*apt*). *Apt* umožňuje prohledávat repozitáře a zjišťovat z nich informace o balíčcích. Na základě těchto informací je možné dalšími nástroji balíčky stahovat, instalovat, aktualizovat, atd.

[16] je nutné důvěřovat veřejnému klíči podepisujícího a mít správně nastavenou politiku - policy

3.2.8 Balíčky RPM

Dalším velmi rozšířeným typem balíčků mezi (nejen) Linuxovými distribucemi je RPM. Tento typ balíčků je používán jako hlavní balíčkovací systém na Linuxových distribucích Red Hat Enterprise Linux, Fedora, aj. Navíc byl také portován na mnoho dalších operačních systémů, např. FreeBSD, Solaris, ... Tento balíčkovací systém byl vyvinut firmou Red Hat a původním význam této zkratky byl Red Hat Package Manager. Nyní je tato zkratka interpretována jako RPM Package Manager.

Stejně jako balíčky DEB, i balíčky RPM se dělí na binární a zdrojové. Binární balíčky mají příponu `.rpm`, zdrojové balíčky `.src.rpm`. Jejich vnitřní struktura se však, na rozdíl od vnitřní struktury balíčků DEB, neliší. Jedná se o binární soubory obsahující hlavičku a cpio archiv.

Zdrojové balíčky obsahují zdrojové kódy programu ve formě, v jaké je poskytl přímo autor programu. Pro vytvoření binárního balíčku tedy může být potřeba provést velké množství kroků. Tyto kroky jsou popsány ve spec souboru (viz níže). Pokud nainstalujeme zdrojový balíček, nedojde k přidání žádného programu, knihovny, ... Abychom mohli nainstalovat samotný program, který zdrojový balíček obsahuje, je nutné jej převést na binární balíček. Ze zdrojového balíčku `balicek.src.rpm` vytvoříme binární balíček `balicek.rpm` pomocí příkladu 3.9.

```
rpmbuild --rebuild balicek.src.rpm
```

Příklad 3.9: Vytvoření balíčku RPM

Soubor spec (resp. `soubor.spec`) je hlavním souborem, který popisuje balíček. Dělí se na hlavičku a sekce. Obsahují vždy název příslušného parametru a jeho hodnotu. Pokud název parametru začíná speciálním symbolem '%', jedná se o víceřádkový textový řetězec, jinak je hodnota parametru jednořádkový textový řetězec.

Hlavička nese informace o balíčku (obdoba souboru `control` u balíčků DEB) [1]:

- Name – název balíčku.
- Version – verze zabaleného programu.
- Release – udává, kolikrát byl daný program zabalen v aktuální verzi.
- Copyright – typ licenčního ujednání.
- Group – specifikuje, do jaké skupiny balíčků má být balíček zařazen.
- Source – webová adresa mainstreamového zdroje nebo informace, který zdrojový balíček odpovídá tomuto binárnímu.
- URL – je obdoba Source, pokud Source obsahuje název zdrojového balíčku, URL obsahuje odkaz na dokumentaci.
- Distribution – linuxová distribuce, pro kterou bylo baleno.
- Vendor – název organizace, která balíček vydává.
- Packager – název organizace, která balíček vytvořila/připravila.

- Requires – definuje, co a v jaké verzi musí systém (prostřednictvím nainstalovaných balíčků) poskytovat, aby bylo možné balíček nainstalovat [17].
- Provides – říká, co balíček poskytuje.
- PreReq – udává, které balíčky musí být nainstalovány před instalací balíčku.
- Conflicts – udává seznam toho, co systém nesmí poskytovat.
- Obsoletes – udává, že balíčky uvedené v tomto seznamu jsou nahrazeny tímto balíčkem, takže je možné provést aktualizaci balíčků na tento balíček, přestože se jmenují jinak.
- %Description – obsahuje popis balíčku.

Sekce:

- %prep – Jedná se o běžný shell skript, který slouží k přípravě prostředí při tvorbě binárního balíčku ze zdrojového balíčku[18].
- %build – Je shell skript, který ze zdrojového balíčku vytvoří binární balíček.
- %install – Je shell skript, který provede instalaci programu.
- %files – Obsahuje seznam souborů, které jsou součástí balíčku. Může specifikovat vlastnosti souborů, jako jsou např. přístupová práva a vlastník.
- %clean – Tato sekce slouží k odstranění souborů, které byly vytvořené při sestavení mimo sestavovací adresář (např. `/tmp`).
- %pre – Shell skript, spuštěný před instalací (obdoba souboru `preinstall` u balíčků DEB).
- %post – Shell skript, který má být spuštěn po instalaci balíčku (obdoba `postinstall`).
- %preun – Shell skript, který bude spuštěn před odinstalací (obdoba `preuninstall`).
- %postun – Shell skript, který má být spuštěn po odinstalaci (obdoba `postuninstall`).
- %Changelog – Seznam změn v programu/balíčku.

Ne všechny sekce musí být vyplněny a použity. Pokud chceme pouze vytvořit binární balíček z již dříve připravených souborů, není nutné používat sekce prep, build, install a clean (tyto sekce se používají při tvorbě binárního balíčku ze zdrojového).

Název výsledného balíčku by měl korespondovat s pravidly pro pojmenovávání balíčku:

```
NázevProgramu-Release-RevizeDistribuce.Architektura.rpm
```

Podepisování balíčku

Stejně jako v případě balíčků DEB je možné i balíček RPM podepsat pomocí GnuPG. Před podepsáním je potřeba nastavit typ podpisu a jméno vlastníka klíče, např. přidáním textu z příkladu 3.10 do souboru `~/.rpmmacros`. Kde JménoVlastníkaKlíčů je jméno, které bylo použito při vytváření GnuPG klíčů [2]. Balíček podepíšeme pomocí příkazu v příkladu 3.11.

[17] neudává, v jakém pořadí se mají provázané balíčky instalovat
[18] obsahuje např. příkazy pro odstranění starých souborů

```
%_signature gpg
%_gpg_name JménoVlastníkaKlíče
```

Příklad 3.10: Typ podpisu a jméno vlastníka

```
rpm --addsign balíček.rpm
```

Příklad 3.11: Podpis balíčku RPM

Repozitáře

I u balíčků RPM se využívají repozitáře. Pro vytváření *yum* (Yellowdog Updater, Modified)[19] repozitářů slouží utilita *createrepo*. Pro vytvoření repozitáře je nutné:

1. Vytvořit HTTP nebo FTP přístupnou adresářovou cestu ve tvaru:

   ```
   */NazevRepozitare/Architektura/
   ```

   Kde NazevRepozitare je koncový adresář, ve kterém se bude repozitář nacházet a Architektura je cílová architektura procesoru, pro kterou jsou dané balíčky sestaveny (např. i386, i686 nebo noarch, pokud jsou pro všechny architektury).

2. Zkopírovat balíčky, které chcete vystavit do adresáře dané architektury.

3. Spustit příkaz pro vytvoření repozitáře v příkladě 3.12.

```
createrepo */NázevRepozitáře/rpms
```

Příklad 3.12: Vytvoření repozitáře RPM

Program *createrepo* vytvoří soubory ve formátu xml. V těchto souborech jsou informace o balíčcích, které repozitář nabízí.

Aby mohl uživatel instalovat programy z nového repozitáře, musí si jej přidat do svého správce RPM balíčků. Pokud uživatel používá *yum*, provede zápis příkazů uvedených v příkladě 3.13 do souboru uloženého standardně na cestě `/etc/yum.repo.d/repo.repo`.

Kde:

- Repo – název sekce;
- name – určuje název repozitáře;
- baseurl – adresa zdroje (může být http, ftp, lokální adresář, ...);
- enabled – povoluje (1) nebo zakazuje (0) používání zdroje správcem balíčků;
- gpgcheck – nařizuje kontrolovat (1) nebo nekontrolovat (0) podpis balíčků;
- gpgkey – určuje, kde se nachází soubor s veřejným klíčem odpovídajícím soukromému, kterým byl balíček podepsán (toto nemusí být specifikováno, pokud je klíč importován pomocí RPM).

Import klíče se provádí pomocí příkazu v příkladu 3.14.

[19] správce RPM balíčků

```
[Repo]
name=My repository
baseurl=http://repo
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY
```

Příklad 3.13: Přidání repozitáře

```
rpm --import RPM-GPG-KEY
```

Příklad 3.14: Import klíče RPM

3.2.9 Balíčky TGZ

Tento druh balíčku je speciálním případem instalačního archivu TGZ z části 3.2.5. Používá se jako jeden z hlavních zdrojů software v operačním systému FreeBSD (a dalších systémech založených na BSD jako je NetBSD, OpenBSD, aj.). Kromě FreeBSD používá tento typ balíčků také Slackware GNU/Linux. Tento typ balíčků slouží k distribuci binárního software. Druhým zdrojem software jsou FreeBSD porty, které se používají pro instalaci software ze zdrojových kódů. TGZ balíček, na rozdíl od TGZ archivu, kromě samotného instalovaného programu a dalších souborů k programu patřících, obsahuje speciální soubory, které slouží i instalaci a informují o obsahu balíčku. Zmíněnými soubory jsou [5]:

- `+COMMENT` – stručný popis balíčku;
- `+CONTENT` – popisuje balíček (jeho název, závislosti na jiných balíčcích) a obsahuje informace o souborech (jejich kontrolní součty a kam je instalovat, resp. odkud je odinstalovat);
- `+DESC` – podrobnější popis zabaleného software;
- `+MTREE_DIRS` – adresářová strukturu instalovaného software.

K nainstalování balíčku slouží příkaz z příkladu 3.15. K odinstalování příkaz z příkladu 3.16.

```
pkg_add NázevBalíčku
```

Příklad 3.15: Instalace balíčku TGZ

```
pkg_delete NázevBalíčku
```

Příklad 3.16: Odinstalace balíčku TGZ

Kapitola 4

Testování

Tato kapitola seznamuje čtenáře s důležitostí testů. Dále uvádí vztah mezi automatickými a ručními testy. Jako zdroj informací v této kapitole byla použita především kniha [9].

Testování SW je zkoumání kvality SW, při kterém se snaží tester získat informace o produktu. Součástí testování je např. vyhledávání chyb v SW systému (případně kontrola, zda byly nahlášené chyby opraveny), hodnocení, nebo porovnávání výkonnosti, bezpečnosti, spolehlivosti, aj. [9]. Alternativní definice říká, že testování SW je proces, procházející všemi fázemi životního cyklu (statickými i dynamickými), zabývající se plánováním, přípravou a hodnocením SW produktů a produktů souvisejících prací. Ve výsledku rozhodne, zda produkty splňují určené požadavky, daný účel a odhalí chyby [3].

4.1 Rozdělení

Existuje mnoho způsobů, jak rozdělit testování. Můžeme jej rozdělit např. podle způsobu provádění, přístupu k testované aplikaci, nebo podle účelu.

Podle způsobu provedení dělíme testování na ruční (manuální) a automatické. Ruční testování provádí tester podle připraveného scénáře. Automatické testování je prováděno automatem a nevyžaduje obvykle zásahy obsluhy.

Dělení testů podle přístupu k testované aplikaci:

- White-box testování – Druh testování, při kterém musí být známé nějaké informace o vnitřním návrhu a chování testovaného SW systému. Příkladem takového testování mohou být unit testy.
- Black-box testování – SW systém je prověřován jako černá skříňka (black-box). Chování SW je ověřováno pomocí využití uživatelského rozhraní a reakcí, které na různá užití uživatelského rozhraní systém vrací. Příkladem takového testu může být test uživatelského rozhraní.
- Gray-box testování – Jedná se o testování, které prověřuje funkcionalitu mezi předchozími dvěma druhy testování. Příkladem takového testu může být nevypsání chybového hlášení při vložení záznamu do databáze, SW systému. White-box test prověřuje, že správně došlo k chybě v případech, kdy záznam nebyl vložen do databáze v příslušné funkční jednotce. White-box testem není možné ověřit, že se o chybě dozvěděl uživatel, protože výpis chyb může být součástí jiné funkční jednotky. Black-box test nemůže tuto chybu také ověřit, protože chybu neočekává, uživatel zadal požadavek a očekává na něj reakci. Touto reakcí ale nemusí být žádný viditelný výsledek. Gray-box test

ověří, zda skutečně došlo k zapsání záznamu do databáze, resp. zda byla vypsána chyba, že k zápisu nedošlo.

Jiným druhem rozdělení je dělení podle účelu (čeho chceme dosáhnout):

- Jednotkové testování (Unit testing) – Slouží k testování funkčních jednotek (unit), jakými jsou například funkce. Provádí se testování na základě známých vstupů a předpokládaných výstupů.
- Regresní testování (Regression testing) – Kontroluje, že funkcionalita z předchozí verze SW systému stále funguje (jsou kontrolovány chyby opravené v předchozích verzích).
- Testování funkcionality (Functional testing) – Kontroluje, zda byla implementována veškerá požadovaná funkcionalita.
- Bezpečnostní testování (Security testing) – Slouží k ověření, zda nejsou přítomna některá z vybraných druhů bezpečnostních rizik. Po analýze zdrojového kódu jsou podezřelá místa prověřena. Tento typ testování slouží obvykle k ověření známých druhů zranitelností, jako je buffer overflow nebo SQL injection.
- Výkonové testování (Performance testing) – Kontroluje, že systém splňuje výkonové požadavky. Obvykle bývá součástí konkurenčního testování.
- Zátěžové testování (Stress testing) – Ověřuje, že SW systém při zátěži funguje správně a nedojde k neočekávanému ukončení.
- Testování spolehlivosti (Soak testing) – Dlouhodobé testování a analýza chování při dlouhodobém běhu. Projevy některých chyb se objevují až při delším běhu programu (nebo jsou tyto problémy výraznější). Test může analyzovat např. kolik paměti program během svého běhu spotřebovává, jestli se časem nezvyšují jeho nároky na procesor, atp.
- Konkurenční testování (Concurrency testing) – Při tomto druhu testování je kladen důraz na analýzu chování při paralelním běhu (např. obsluha více požadavků současně).
- Testování pokrytí kódu (Code coverage testing) – Měří kolik procent kódu bylo pokryto (vykonáno) při použití testovací sady.
- Testování interoperability (Interoperability testing) – Prověřuje spolupráci různých komponent. Používá se na příklad, pokud má komponenta vyhovovat komunikačnímu standardu.

4.2 Automatizace testování

Testování SW produktu je důležitým krokem, který by měl proběhnout, má-li být odstraněno co nejvíce chyb, které byly do programu zaneseny během vývoje nebo sestavování. Testy by měly pokrýt jak základní funkcionalitu, tak pokročilejší funkce. Testování, jak již bylo napsáno v předchozí části, může probíhat manuálně podle předem připraveného scénáře. Ruční testování má ovšem mnoho negativních vlastností:

- Je časově náročné, protože člověk je obvykle pomalejší při provádění některých operací než skript/program.
- Je náchylné na chyby obsluhy, protože automatický test se vždy přesně drží scénáře, člověk může snadněji pochybit (ať už z důvodu nepozornosti, únavy, ...).
- Zahrne menší zlomek konfigurací, ať už hardware nebo kombinací nainstalovaného software (souvisí s časovou náročnosti).
- Je problematicky opakovatelné (souvisí s náchylností na chyby obsluhy).

Na rozdíl od manuálního, automatické testování vede k redukci nákladů, času a potřebných zdrojů. Automatickým testováním nelze jednoduše nahradit manuální. Oba způsoby by se spíše měly vzájemně doplňovat a kombinovat. Existují speciální druhy testů, které jsou ručním testováním komplikovaně proveditelné a chyby proto špatně odhalitelné. Příkladem těchto druhů testů je detekce neuvolňování paměti a neoprávněný přístup do paměti. Některé testy nelze automatizovat, např. test uživatelské přívětivosti SW systému může posoudit pouze několik vzorků z cílené skupiny potenciálních zákazníků.

Jak již bylo zmíněno, ne vše je možné automatizovat. Podle analýzy uvedené v knize [9] je možné automatizovat od 40 % do 60 % testů. Kniha se také odkazuje na dotazník ze semináře „Automated testing selected best practices“. Na otázky v dotazníku odpovídalo přes 700 účastníků, z čehož 40 % uvedlo, že jejich společnost má méně než 300 zaměstnanců a 60 % odpovědělo, že má více než 300 zaměstnanců. Jedním z dotazů, bylo, jak velkou část z rozpočtu společnosti utratí za testování. Zde 46 % dotázaných uvedlo, že utratí 30 % až 50 % rozpočtu, a dalších 19 % uvedlo, že je to od 50 % do 75 %. Součástí dotazníků byl i dotaz proč automatizace selhává. Uvedené důvody byly:

- 37 % – nedostatek času;
- 17 % – nedostatečný rozpočet;
- 11 % – nekompatibilní nástroje;
- 20 % – nedostačující odbornost;
- 15 % – jiné (kombinace výše zmíněných atp.).

Ne vždy se vyplatí automatizovat. Jednorázové ruční testování je vždy levnější než výroba automatizovaného testu. Ruční test vyžaduje stejnou analýzu a přípravu scénáře, ale na rozdíl od automatizovaného testu nevyžaduje implementaci a je možné jej jednorázově provést (pokud to charakter testu umožňuje). Okamžité provedení ušetří čas a finance za vývoj automatizovaného testu. Proto je nutné zvážit, které testy chceme implementovat automatizovaně. V tomto rozhodování by měla pomoci analýza návratnosti investicí (Return Of Investments). Existuje několik druhů těchto analýz. Jedna z nich je popsána v [9] a bude použita pro účely této práce.

Hlavními přínosy automatizace testů jsou:

- Automatická verifikace správného sestavení SW systému (Smoke testy[1]) – Pro tento účel se používá sada testů základní funkcionality SW systému. Tato sada by měla ověřit, že nedošlo k nějaké zásadní chybě, kvůli které nemá smysl spouštět pokročilejší testy.

[1] název původně pochází z elektroniky, kdy je zapojení elektrického obvodu ověřeno připojením napětí a pokud se z obvodu nezačne kouřit (smoke), má smysl jej dále testovat

- Vylepšené regresní testování – regresní testy by měly podrobně porovnat funkcionalitu systému mezi jeho dvěma verzemi a tím ověřit, že je funkcionalita systému i po opravě chyby zachována.
- Multiplatformní testy a testování na různých konfiguracích – s použitím vhodných virtualizačních nástrojů může být systém prověřován na větším množství konfigurací cílového systému a to jak z hlediska HW i SW.
- Vylepšené spouštění triviálních testů – při testování většiny SW systémů se objevují testy, které jsou neustále opakovány, a proto náchylné k chybám. Tato náchylnost plyne z monotónnosti dané práce, díky čemuž může být pracovník unaven a nemusí být tak pozorný, jak by bylo potřeba. Automatizovaný skript nebude nikdy unaven.
- Výhodnost tvorby pokročilých testů – protože je možné testy opakovat, je výhodné zaměřit se i na druhy testů, které jsou složitě proveditelné, nebo vyžadují větší množství času.
- Testování částí, které ruční testování nemůže efektivně zachytit – některé druhy testů jsou ručně testovatelné špatně, nebo jsou netestovatelné.
- Schopnost reprodukovat chyby – jedním z problémů, se kterými se testovací oddělení setkávají je, že po nalezení chyby nejsou schopna tuto chybu znovu navodit. Automatické testy tento problém řeší, protože se řídí přesně daným naskriptovaným chováním, které je možné kdykoliv libovolně zopakovat.
- Získání expertního systému – pokud při testování dojde k nepřítomnosti některého z klíčových členů testovacího týmu, může dojít ke zdržení, protože danou část systému prověřoval chybějící člen. Pokud je používáno automatizované testování, tak i v případě nepřítomnosti některých členů týmů, mohou automatizované testy proběhnout.
- Testování mimo pracovní dobu – protože automatizované testy nevyžadují interakci s uživatelem, je možné testování naplánovat na kteroukoliv denní nebo noční hodinu. Toto umožňuje testy naplánovat i mimo běžnou pracovní dobu. Díky tomu mohou testeři při nástupu do práce zkontrolovat, jak dopadly testy naplánované předešlý den. Navíc při vhodném plánování může být použit HW zaměstnanců, kteří zrovna nepracují.
- Vylepšené testování výkonu – automatizované testy mohou zaznamenávat události a z nich zpětně vyhodnotit, jak roste, nebo klesá výkonnost systému.
- Vylepšené zátěžové testování – tyto testy jsou při ručním testování problematické, protože je potřeba systém velmi zatížit (obvykle z více počítačů). Vytvoření velkého zatížení systému vyžaduje koordinaci, která nebývá při ručním testování dostatečná. Z tohoto důvodu je ruční zátěžové testování problematicky reprodukovatelné. Při automatickém testování tuto koordinaci zajišťuje testovací nástroj a ten také usnadňuje opakovatelnost testu.
- Kvalitativní měření a testy optimalizace – protože scénáře automatických testů můžeme opakovaně spouštět, je možné z výstupů testů hodnotit, zda klesají nebo stoupají nároky SW systému, a jak se případně vyvíjí výkon takového systému.

- Vylepšení vývojového cyklu systému – existují nástroje pro automatické testování, které jsou stavěné na cykly vývoje SW. Tyto nástroje se snaží využívat jednotlivých fází vývoje SW k přípravě automatických testů.
- Vylepšená dokumentace a sledování – automatizované testy obvykle ukládají své konfigurace a výsledky testů do nějaké formy databáze, díky čemuž je možné vyhodnocovat a sledovat průběh vývoje systému.
- Distribuce zátěže a konkurenční testování – při používání virtualizačních nástrojů je možné distribuovat zátěž různých testů na více počítačů. Navíc může být použita pro některé testy síťových produktů i firemní síť pro nasimulování reálných situací.

Kapitola 5

Realizace

V části 5.1 tato kapitola popisuje překládací a sestavovací proces AVG. Je zde popsáno, jak fungovalo překládání a sestavení ve verzi AVG for Linux/FreeBSD 8.5, co bylo v nové verzi AVG for Linux/FreeBSD 2011 vylepšeno a co tato vylepšení přinesla.

V následující části (5.2) je popsán testovací nástroj a také je uvedeno, proč byl tento nástroj vyvinut a jak funguje.

V poslední části 5.3 jsou popsány testy, které byly navrženy, implementovány a použity pro testování funkcionality AVG.

5.1 Sestavení

Proces automatického překladu a sestavení, jak byl popsán v kapitole 3.1, nebyl u produktu AVG plně implementován. Součástí této práce bylo navrhnout a implementovat navržená zlepšení tak, aby došlo k zefektivnění celého procesu.

5.1.1 Původní stav AVG for Linux/FreeBSD 8.5

AVG for Linux/FreeBSD 8.5 je sestavováno pomocí GNU/Make. Protože jsou tyto OS rozdílné, provádí se nastavení některých parametrů překladu podmíněně. Vychází z podobného principu, jaký používají nástroje ze skupiny GNU Autotools (popsané v části 3.1.4), tzn. před samotným překladem jsou nastaveny platformě závislé proměnné, které jsou potom využívány při překladu. Protože se provádí překlad na Unixových systémech, není nutné řešit problémy, které by mohly vzniknout při pokusu o překlad na OS, který není POSIX kompatibilní (např. neexistence příkazů shellu, které POSIX definuje). Překlad, sestavení, testování a zveřejnění probíhá v následujících krocích:

1. Vytvoření seznamu změněných souborů od poslední vydané verze AVG.
2. Sestavení AVG ze zdrojových textů (bez umožnění paralelního běhu).
3. Rozbalení poslední vydané verze AVG.
4. Vytvoření archivu TGZ z nezměněných souborů (z kroku 3) a změněných souborů (z kroku 2) podle seznamu z kroku 1 a doplnění poslední verze virové databáze.
5. Ruční vytvoření konfiguračního souboru AVG a verzí komponent do archivu TGZ.

6. Automatické sestavení Full distribučních balíčků a archívu (SH, DEB, RPM) z archivu TGZ vytvořeného v krocích 4 a 5.
7. Ruční sestavení Free archivů a balíčků z distribučních balíčků.
8. Asistované vygenerování aktualizací.
9. Ruční testování shodnosti binárního obsahu archivů a balíčků oddělením QA.
10. Ruční testování nahlášených chyb oddělením QA.
11. Zveřejnění.

Jak je z postupu vidět, některé kroky sestavení jsou prováděny ručně nebo asistovaně (kroky 5, 7 a 8). Nutnost ručních zásahů vyžaduje kontrolu, zda daná fáze již skončila a dává prostor k chybám. V kroku 2 dochází k sestavení celého produktu ze zdrojových kódů. Toto je zbytečné, protože ve fázi slučování se starým balíčkem (krok 4) jsou použity pouze některé soubory. Balíčky jsou předávány bez základní kontroly dále do oddělení QA, které provede ruční ověření binární shodnosti. To znamená, že v případě nalezení chyby je zde prodleva mezi sestavením, zjištěním chyby, odstraněním chyby a testováním.

5.1.2 Aktuální stav AVG for Linux/FreeBSD 2011

Překlad nové verze AVG for Linux/FreeBSD 2011 je také sestavován pomocí GNU/Make. Z důvodů popsaných v části 5.1.1 nebylo žádoucí jej měnit. Makefile pro překlad byly upraveny tak, že překlad nyní probíhá v těchto krocích:

1. Vytvoření seznamu změněných souborů od poslední vydané verze AVG.
2. Sestavení změněných souborů AVG ze zdrojových textů (umožňuje paralelizaci).
3. Rozbalení poslední vydané verze AVG.
4. Vytvoření archivu TGZ z nezměněných souborů (z kroku 3) a změněných souborů (z kroku 2) podle seznamu vytvořeného v kroku 1 a doplnění poslední verze virové databáze.
5. Automatické vytvoření konfiguračního souboru AVG a verzí komponent do archivu TGZ.
6. Automatické sestavení distribučních balíčků a archívu (SH, DEB, RPM, TGZ_FREE, SH_FREE, DEB_FREE, RPM_FREE) z archivu TGZ vytvořeného v krocích 4 a 5.
7. Automatické vygenerování aktualizací.
8. Automatické testování správného sestavení pomocí testů popsaných v části 5.3.
9. Ruční testování nahlášených chyb oddělením QA.
10. Zveřejnění.

Při porovnání seznamu v části 5.1.2 se seznamem v části 5.1.1 zjistíme, že hlavní rozdíl je v automatizaci kroků 5, 7 a sestavení Free balíčků. Automatizovaným překladem je zmenšen prostor pro chyby, které může obsluha udělat. Díky automatizaci je překlad možné provádět i bez hlubší znalosti celého sestavovacího procesu. Dále byla omezena množina překládaných souborů na nutné minimum (v kroku 2). Překládány jsou pouze soubory, které mají být skutečně součástí výsledného balíčku. Tento krok přináší výraznou úsporu času, protože při sestavení nové verze distribučních balíčků obvykle nedochází k nahrazení většiny souborů, které mohou být převzaty ze starší verzí balíčků. Makefile byl upraven, aby umožňoval spuštění paralelního překladu. Porovnání rychlostí překladů Linuxové verze AVG je možné vidět v tabulce 5.1. Z této tabulky vyplává, že překlad nové verze AVG 2011 je o $6,34$ % pomalejší než překlad předešlé verze 8.5. Toto je pravděpodobně způsobeno překladem nových nástrojů, které nebyly součástí předchozí verze. Z tabulky také vyplývá, že při paralelním překladu je nová verze při využití čtyř paralelně zpracovaných úkolů rychlejší o $223,64$ % a u osmi paralelně zpracovávaných úkolů o $285,32$ % než původní verze. Způsob měření je popsán v příloze A.

Verze AVG for Linux	1 paralelní zpracování [s]	4 paralelní zpracování [s]	8 paralelních zpracování [s]
8.5	$1549,81 \pm 5,85$	#	#
2011	$1654,82 \pm 15,88$	$478,86 \pm 9,3$	$402,21 \pm 3,95$

Tabulka 5.1: Rychlost překladu a sestavení různých verzí AVG for Linux

Symbolem # jsou označeny testy, které nejsou u dané verze produktu podporovány.

V tomto měření nebyla porovnávána rychlost vytvoření výsledných distribučních balíčků a souborů pro aktualizace. Porovnání nemohlo být provedeno, protože sestavovací proces AVG 8.5 zahrnoval ruční kroky, které by měření znehodnotily. Manuální kroky, které je nutné provést pro sestavení balíčků AVG 8.5 a jeho aktualizací, trvají přibližně 40 minut. Automatizovaný systém použitý v AVG 2011 provede tyto kroky přibližně za 5 minut.

5.1.3 Balíčky

AVG ve verzi 8.5 používá čtyři druhy (resp. osm druhů, počítáme-li i Free balíčky jako samostatné) instalátorů pro Linux a jeden (resp. dva) pro FreeBSD. Množství ani druh instalátorů nebyl pro novou verzi AVG 2011 změněn, protože analýzou využívanosti balíčků na různých distribucích Linuxu bylo zjištěno, že RPM a DEB balíčky jsou majoritní a archivy TGZ a SH pokrývají zbylé distribuce. Analýza využívanosti různých typů balíčků je probírána v části 3.2.3.

Analýzou balíčků AVG ve verzi 8.5 bylo zjištěno, že:

- Plně nedodržují požadavky a doporučení, které jsou specifikovány v definicích balíčků.
- Nejsou podepisovány, a proto jsou podvrhnutelné.

Bylo navrženo řešení některých zmíněných problémů. Ve verzi AVG 2011 by měly být některé ze zmíněných nedostatků opraveny.

5.2 Testovací nástroj

V kapitole 2 je už zmíněno, že větší část funkcionality AVG je zaměřena na hrozby přicházející z počítačové sítě. Abychom tuto funkcionalitu důkladně ověřili, je nutné testovat chování AVG v počítačové síti. Testy v počítačové síti probíhají na firemních serverech, na které je duplikována běžná pošta AVG. Jsou to počítače vyhrazené pro tento účel. Díky virtualizačním nástrojům, jako je VirtualBox [38] nebo VMware Workstation [25], můžeme simulovat chování počítačů s různou konfigurací HW a různými OS na jednom fyzickém počítači. Ve firmě AVG Technologies CZ, s. r. o. jsou k tomuto účelu využívány nástroje firmy VMware.

Aby mohla probíhat simulace počítače automaticky, je vhodné použít nástroj pro automatizaci testů jako je STAF/STAX (Software Testing Automation Framework) [37]. Tento nástroj umožňuje automatizované testování pomocí naskriptovaného chování počítače a následného kontrolování výsledků jednotlivých testů. STAF umožňuje řídit chování jednoho počítače, ale neumožňuje simulovat chování počítačů ve virtuální síti.

Protože nebyl nalezen vhodný nástroj, který by umožňoval řídit chod počítačů ve virtuální síti tvořené virtuálními stroji ve VMware Workstation, byl pro tento účel navržen a implementován nový nástroj.

5.2.1 Vlastnosti nového nástroje

Při návrhu byly specifikovány tyto hlavní požadavky:

- kopírování souborů obousměrně;
- spouštění programů ve virtuálních strojích;
- multiplatformnost;
- propojování virtuálních strojů pomocí sítě;
- možnost synchronizace virtuálních strojů.

Pro splnění těchto cílů byl nástroj naprogramován v jazyce C++. Komunikuje s VMware pomocí VIX API. Toto API je vývojářské rozhraní, pomocí kterého VMware komunikuje s jinými aplikacemi. API je k dispozici pro jazyky C, Perl a COM (Visual Basic, VBscript, C#). Je určeno pro psaní programů a skriptů, které automatizují běh virtuálních strojů (spouštění programů, manipulace se soubory, atp.) [24]. Pomocí tohoto API je možné spouštět na virtuálních strojích programy a skripty, kopírovat soubory, ...

Program je multiplatformní. Při psaní programu byla použita knihovna Qt [35], která je k dispozici pro velké množství platforem. Jiná nesystémová knihovna při vývoji použita nebyla.

Nástroj také umí měnit topologii sítě virtuálních strojů. Změna topologie se provádí přidáváním a odebíráním síťových karet na jednotlivých virtuálních strojích. Každá síťová karta je připojena do virtuálního switche, což je rozhraní, které specifikuje, jaké síťové karty spolu mohou komunikovat. Přidávání síťových karet a jejich zařazování do switche je prováděno úpravou konfiguračního souboru virtuálního stroje (`soubor.vmx`).

Na obrázku v příloze B.1 je znázorněn diagram tříd. Díky čistě virtuálním rozhraním Command (pro příkazy probíhající na hostiteli), HostCommand (pro příkazy ovládající virtuální stroj) a VmCommand (sloužící pro spouštění příkazů ve virtuálním stroji) je možné přidávat další příkazy s minimem zásahů do jiných tříd. Jediným dalším nutným zásahem

je úprava v metodě load třídy CommandInterpreter, která zajišťuje načítání dat z konfiguračního souboru. V diagramu je možné také vidět virtuální třídu Thread, která předává vlastnost spuštění ve vlastním vláknu svým potomkům. Této vlastnosti využívají třídy CommandInterpreter a VirtualMachine. Z diagramu nevyplývá účel třídy SyncPoint. Třída slouží k synchronizaci instancí třídy CommandInterpreter, které jsou spuštěny každá ve vlastním vláknu. SyncPoint je singelton [27], který využívá příkaz/třída Sync. Při zavolání metody addSyncPoint je přičteno jedna k počítadlu vláken, na která se má čekat. Před spuštěním simulace je zavolána metoda init třídy SyncPoint, která inicializuje pthread_barrier_t s počtem vláken, na která se má čekat. Při zavolání metody run třídy Sync je zavolána metoda reached třídy SyncPoint. Tato metoda obsahuje jediný příkaz pthread_barrier_wait, který vlákno uspí (pokud nebylo dosaženo počtu čekajících vláken), nebo všechna vlákna probudí (pokud čeká už dostatečný počet vláken).

K načítání dat ze souboru a jejich extrakci z formátu XML je použita část knihovny Qt.

5.2.2 Vytvoření scénáře a topologie

Vyvinutý nástroj používá ke svému nastavení jeden konfigurační soubor ve formátu XML pro jeden scénář. Soubor popisuje topologii virtuální sítě a chování jednotlivých virtuálních strojů.

K nastavení topologie virtuální sítě slouží v konfiguračním souboru sekce:

- VirtualNetwork – označuje síť virtuálních strojů. Může obsahovat 0–n VirtualMachine:
  - VirtualMachine – virtuální stroj. Může obsahovat 0–n NetworkInterface:
    - NetworkInterface – virtuální síťová karta. Má jeden parametr:
      - Device – zařízení VMware, které má virtuální síťová karta využít.

Chování jednotlivých virtuálních strojů je popsáno sekcí VirtualMachine pomocí interpreta příkazů:

- CommandInterpreter – Interpret příkazů je vykonavatelem větve příkazů. Každý virtuální stroj musí mít právě jeden CommandInterpreter. Dále, každá větev vytvořená příkazem Fork (viz níže), musí být uzavřena ve vlastní sekci CommandInterpreter. CommandInterpreter má jeden parametr:
  - Id – Je jednoznačný identifikátor, který se používá při slučování větví příkazů. Id CommandInterpretera odpovídá parametr CommandInterpreter příkazu Join.

CommandInterpreter může vykonávat tyto příkazy:

- ConnectHost – připojí se k hostiteli (musí být na začátku každého skriptu);
- CopyFileFromGuestToHost – zkopíruje soubor z virtuálního stroje na hostitelský:
  - GuestFilePath – cesta k souboru ve virtuálním stroji;
  - HostFilePath – cesta, kam má být na hostitelský stroj soubor nahrán (včetně výsledného jména souboru);
- CopyFileFromHostToGuest – zkopíruje soubor z hostitelského stroje na virtuální;

  - GuestFilePath – cesta, kam má být na virtuální stroj soubor nahrán (včetně výsledného jména souboru);
  - HostFilePath – cesta k souboru na disku hostitelského stroje;
- CreateSnapshot – vytvoří nový snapshot[1];
- Echo – vypíše zprávu do konzole na hostitelském počítači. Má jeden parametr:
  - Value – textový řetězec, který má být vypsán;
- Fork – vytvoří novou větev příkazů, zpracovávanou paralelně se stávající větví;
- Join – místo, kde má dojít ke spojení větví příkazů. Má parametr:
  - CommandInterpreter – označuje interpreta, na kterého se má počkat;
- Login – přihlásí uživatele k virtuálnímu stroji. Tento příkaz musí předcházet příkazy Logout, WriteVariable, ProgramInGuest, ScriptInGuest, CopyFileFromHostToGuest, CopyFileFromGuestToHost. Má dva parametry:
  - Login – přihlašovací jméno;
  - Password – heslo;
- Logout – odhlásí uživatele od virtuálního stroje (musí předcházet každému novému přihlášení jiného uživatele);
- PowerOff – vypne virtuální stroj;
- PowerOn – zapne virtuální stroj;
- ProgramInGuest – spustí ve virtuálním stroji program s parametry. Parametry jsou:
  - Program – program, který má být spuštěn (včetně plné cesty);
  - Params – parametry, které mají být programu předány;
- RemoveCurrentSnapshot – odstraní aktuální snapshot;
- RevertCurrentSnapshot – vrátí se ke stavu uloženém v posledním snapshotu;
- ScriptInGuest – spustí ve virtuálním stroji skript. Parametry:
  - Interpreter – interpret skriptovacího jazyka;
  - Script – skript, který má být spuštěn;
- Sync – označuje místo skriptu, kde se má počkat na některý s dalších virtuálních strojů. Má parametr:
  - Point – číslo sloužící k identifikaci synchronizačního bodu. Čeká se vždy, dokud všechny stroje nedorazí k bodu s daným číslem (pokud takový bod mají);

[1] snímek aktuálního stavu virtuálního stroje VMware. Uloží všechny informace, které jsou potřeba pro obnovení ukládaného stavu virtuálního stroje, jako je obsah paměti, obraz změn na disku, spuštěné programy, atd.

- WriteVariable – zapíše hodnotu proměnné do prostředí virtuálního stroje. Parametry jsou:
  - Variable – jméno proměnné;
  - Value – hodnota proměnné.

5.2.3 Použití nástroje

Program se spouští s jedním povinným parametrem:

- -l `soubor` – pro načtení dat ze `souboru`.

Po spuštění program načte ze `souboru` konfiguraci a začne ji provádět. Před provedením nějakého příkazu je vypsáno na standardní výstup hlášení ve tvaru:

```
RidiciSoubor CommandInterpretter CisloInterpreta command CisloPrikazu: Hlaseni
```

Kde:

- RidiciSoubor – Je cesta ke konfiguračnímu souboru vmx virtuálního stroje VMware. Slouží k identifikaci konkrétního stroje při paralelním běhu.
- CisloInterpreta – Číslo interpreta, který daný příkaz provádí. Pomocí tohoto čísla je, spolu s číslem příkazu, možné identifikovat příkaz, který selhal.
- CisloPrikazu – Pořadové číslo právě prováděného příkazu.
- Hlaseni – Informuje uživatele o právě probíhající akci.

Při spuštění programu se skriptem uvedeným v příkladu 5.1 bude program provádět tyto operace:

1. Načte konfiguraci z XML souboru.
2. Upraví se síťové karty v souboru `./Debian6-server.vmx` tak, že se odeberou všechny přítomné a přidá se jedna nová připojená do switche `/dev/vmnet1`, podle řádku 3 skriptu.
3. Spustí se CommandInterpreter v novém vláknu, který začne provádět příkazy:
   - řádek 5 – připojí se k VMware;
   - řádek 6 – spustí virtuální stroj Debian6-server;
   - řádek 7 – přihlásí se k virtuálnímu stroji jako uživatel root;
   - řádek 8 – spustí ve virtuálním stroji skript, který zapíše „Hello world“ do souboru `/tmp/out`;
   - řádek 9 – zkopíruje soubor z virtuálního stroje do aktuálního adresáře;
   - řádek 10 – odhlásí uživatele root;
   - řádek 11 – vypne virtuální stroj.
4. Program skončí.

Výstupem bude:

- Výpis na standardní výstup. Tento výstup je v příkladu 5.2.
- Soubor `out` zkopírovaný do adresáře ./ na hostiteli.

```
1: <VirtualNetwork>
2:     <VirtualMachine VmxFile=,,./Debian6-server.vmx‘‘>
3:         <NetworkInterface Device=,,/dev/vmnet1‘‘/>
4:         <CommandInterpreter Id=,,0‘‘>
5:             <ConnectHost/>
6:             <PowerOn/>
7:             <Login Login=,,root‘‘ Password=,,root‘‘/>
8:             <ScriptInGuest Script=,,echo Hello world > /tmp/out‘‘ Interpreter=,,/bin/bash‘‘/>
9:             <CopyFileFromGuestToHost HostFilePath=,,./out‘‘ GuestFilePath=,,/tmp/out‘‘/>
10:            <Logout/>
11:            <PowerOff/>
12:        </CommandInterpreter>
13:    </VirtualMachine>
14:</VirtualNetwork>
```

Příklad 5.1: Skript testovacího nástroje

```
/vmware/Debian6-server/Debian6-server.vmx CommandInterpretter 1 command 0: connecting host
/vmware/Debian6-server/Debian6-server.vmx CommandInterpretter 1 command 1: powering on
/vmware/Debian6-server/Debian6-server.vmx CommandInterpretter 1 command 2: logging in as user: root
/vmware/Debian6-server/Debian6-server.vmx CommandInterpretter 1 command 3: running in guest script echo
Hello world! > /tmp/out with interpreter /bin/bash
/vmware/Debian6-server/Debian6-server.vmx CommandInterpretter 1 command 4: coping file /tmp/out from
guest to host ./out
/vmware/Debian6-server/Debian6-server.vmx CommandInterpretter 1 command 4: failed
/vmware/Debian6-server/Debian6-server.vmx CommandInterpretter 1 command 5: logging out
/vmware/Debian6-server/Debian6-server.vmx CommandInterpretter 1 command 6: powering off
```

Příklad 5.2: Standardní výstup spuštění skriptu 5.1

Hlavní přínosy nástroje jsou:

- Schopnost přidávat a odebírat síťové karty virtuálních strojů.
- Spouštění, vypínání a synchronizace virtuálních strojů.
- Spouštění paralelních příkazů na virtuálních strojích.
- Vytváření, vracení se a odstraňování snapshotů.
- Kopírování souborů skrze VIX API. Na rozdíl od nástrojů jako STAF/STAX, nepoužívá implementovaný nástroj ke kopírování počítačovou síť. Proto kopírování funguje i v případech, kdy síť není dostupná nebo nakonfigurovaná.

5.3 Testování

Pro kontrolu funkcionality AVG bylo navrženo několik testů. Tyto testy mají charakter Black-box testování. Pro testování byl použit nástroj k tomuto účelu vyvinutý. Nástroj je blíže popsaný v části 5.2.

Testy jsou připraveny ve formě šablon a shell skriptů. Shell skript slouží ke komunikaci s uživatelem. Provede načtení parametrů, jejich kontrolu a následně, na základě těchto parametrů, provádí substituci proměnných v šabloně na konkrétní hodnoty. Výsledkem tohoto nahrazení je XML soubor popisující chování virtuálních strojů pro testovací nástroj. Shell skript zavolá nástroj a po jeho ukončení vymaže vygenerovanou konfiguraci.

Byly implementovány tyto testy:

1. Smoke tests – Sada testů, které ověřují základní funkcionalitu. Konkrétně:
   (a) instalovatelnost balíčku,
   (b) registrovatelnost balíčku v licenci TRIAL/FREE/FULL,
   (c) spuštěnost jednotlivých komponent,
   (d) seznam zastavených komponent,
   (e) seznam komponent v chybovém stavu,
   (f) scan souboru,
   (g) připojení k avgtcpd,
   (h) požadavek na scan souboru skrze avgtcpd,
   (i) odinstalovatelnost balíčku.

   Návrh testu je na obrázku 5.1.

   Vstupy:

   - balíček, který má být otestován,
   - cesta k vmx souboru virtuálního stroje,
   - bash příkaz k instalaci balíčku,
   - cesta, kam má být uložen výsledek testu (včetně názvu souboru).

   Výstup:

   - Soubor, který obsahuje výpis jednotlivých částí testu se stavem yes/no podle toho, zda se povedlo nebo nepovedlo danou akci uskutečnit.

2. Test avgavid – Tento test ověřuje:
   (a) Vytvoření virtuální paměti – Test změní maximální velikost sdílené paměti tak, aby se do ní vešla virová databáze. Dále test ověří že avgavid sdílenou paměť alokovalo.
   (b) Připojování procesů scand k virtuální paměti – Tato část testu požádá o několik skenů adresáře / a zkontroluje, zda se příslušné množství procesů připojilo ke sdílené paměti.

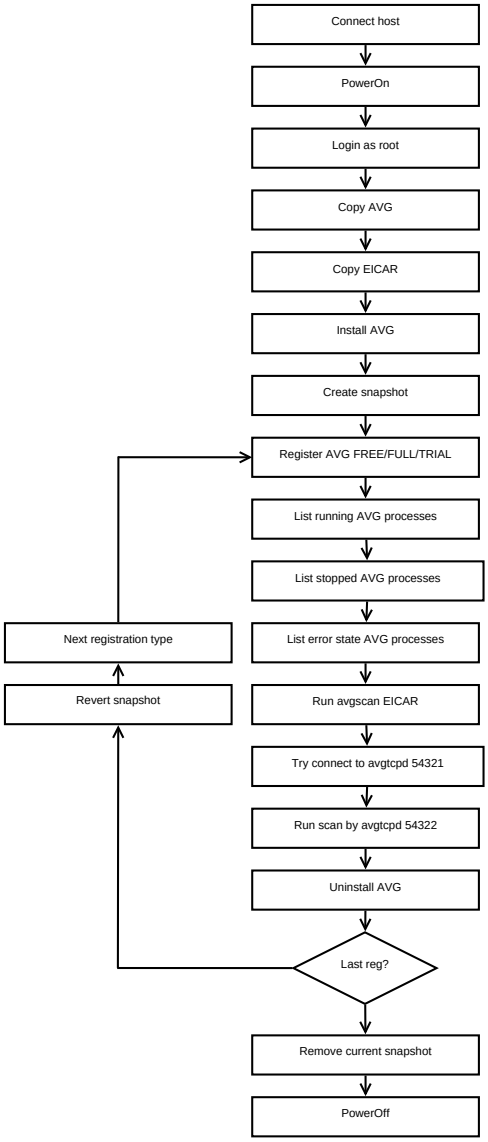
   Vývojový diagram viz na obrázku 5.2.

   Vstupy:

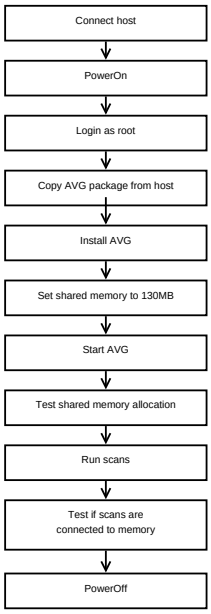
   - balíček k otestování,
   - cesta k vmx souboru virtuálního stroje,
   - bash příkaz, pomocí kterého je možné balíček nainstalovat,
   - cesta, kam má být uložen výsledek testu (včetně názvu souboru).

   Výstup:

   - Soubor jednotlivých testovacích částí se stavem yes/no podle toho, zda se povedlo nebo nepovedlo danou akci uskutečnit.



Obrázek 5.1: Vývojový diagram smoke testů



Obrázek 5.2: Vývojový diagram testu avgavid

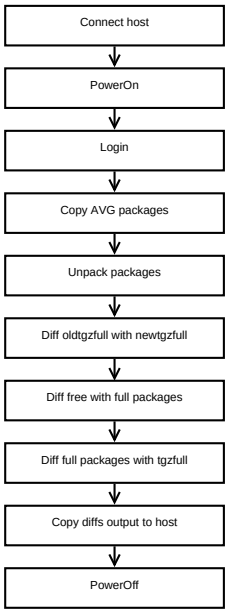
3. Test binární shodnosti balíčků – Test ověřuje, že se obsah nově sestavených balíčků mezi sebou neliší a kontroluje, zda byly zabaleny pouze změněné soubory, které byly požadovány. Průběh testu je vidět na obrázku 5.3.

   Vstupy:

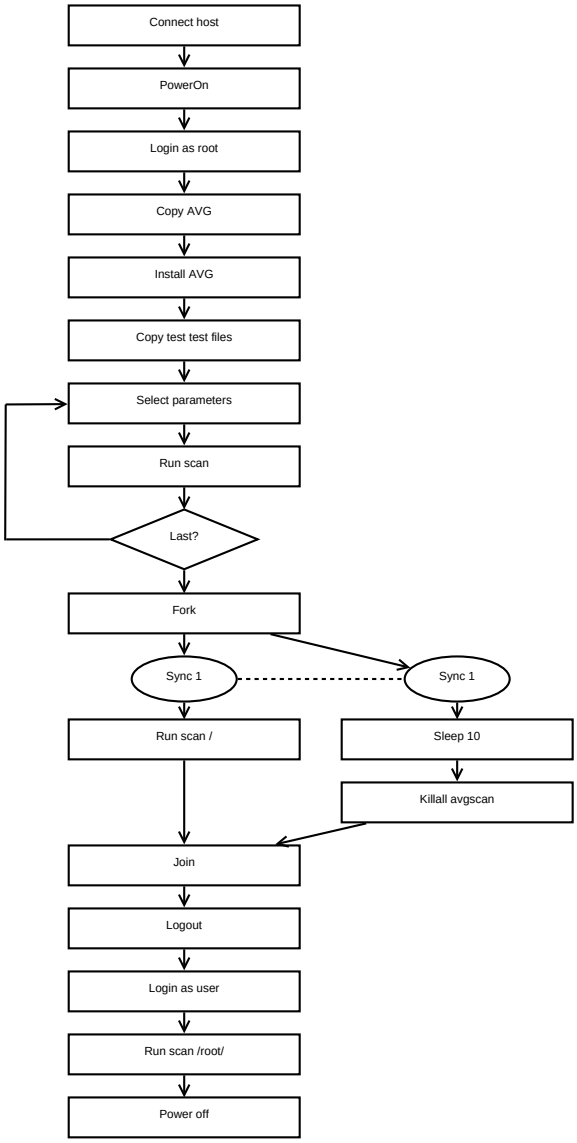
   - sada nově sestavených balíčků,
   - balíček TGZ, který byl sestaven při předchozím sestavovacím cyklu.

   Výstup:

   - Soubor s výstupem rozdílů adresářů porovnaných programem `diff`.



Obrázek 5.3: Vývojový diagram testu binární shodnosti balíčků



Obrázek 5.4: Vývojový diagram testu návratových hodnot avgscan

4. Test parametrů avgscan – Tento test kontroluje detekční schopnost a návratové hodnoty avgscan. Kontrola je prováděna při opakovaných bězích a je kontrolována návratová hodnota, která by měla odpovídat hodnotám uvedeným v manuálových stránkách. Vývojový diagram tohoto testu je na obrázku 5.4.

   Vstupy:

   - balíček s AVG,
   - příkaz, pomocí kterého je možné AVG nainstalovat,
   - cesta k vmx souboru virtuálního stroje,
   - cesta, kam má být uložen výsledek testu.

Výstup:

- Soubor s výsledky testů.

5. Test detekčních schopností – Tento test slouží k ověření AVG při kontrole emailové pošty. Při testu se kontroluje, zda AVG detekuje při různých nastaveních hrozby, které přicházejí jako součást emailů správně. Test je znázorněn na vývojovém diagramu 5.5.

   Vstupy:

   - balíček DEB AVG,
   - příkaz, pomocí kterého je možné AVG nainstalovat,
   - licenční číslo, pomocí kterého je možné nainstalované AVG registrovat,
   - cesta k vmx souboru virtuálního stroje, který bude sloužit jako server,
   - cesta k vmx souboru virtuálního stroje, sloužícího jako klient,
   - adresář s testovací sadou emailů,
   - adresář se sadou různých konfigurací AVG, které budou postupně aplikovány,
   - soubor, kam mají být ukládány výsledky testů.

   Výstup:

   - Informace, zda obsah emailových schránek, který je zkopírován při každé konfiguraci, odpovídá referenční sadě.

6. Test avgupdate a správného vytvoření aktualizačních souborů – Tento test kontroluje schopnost aktualizovat starou verzi AVG na novou. Kontroluje se:

   - Schopnost aktualizovat se na novou verzi pomocí rozdílových aktualizačních souborů.
   - Schopnost aktualizovat se na novou verzi pomocí aktualizačních souborů s plným obsahem.
   - Rozdíl mezi aktualizovanou verzí ze starého balíčku a novou verzí AVG z balíčku.

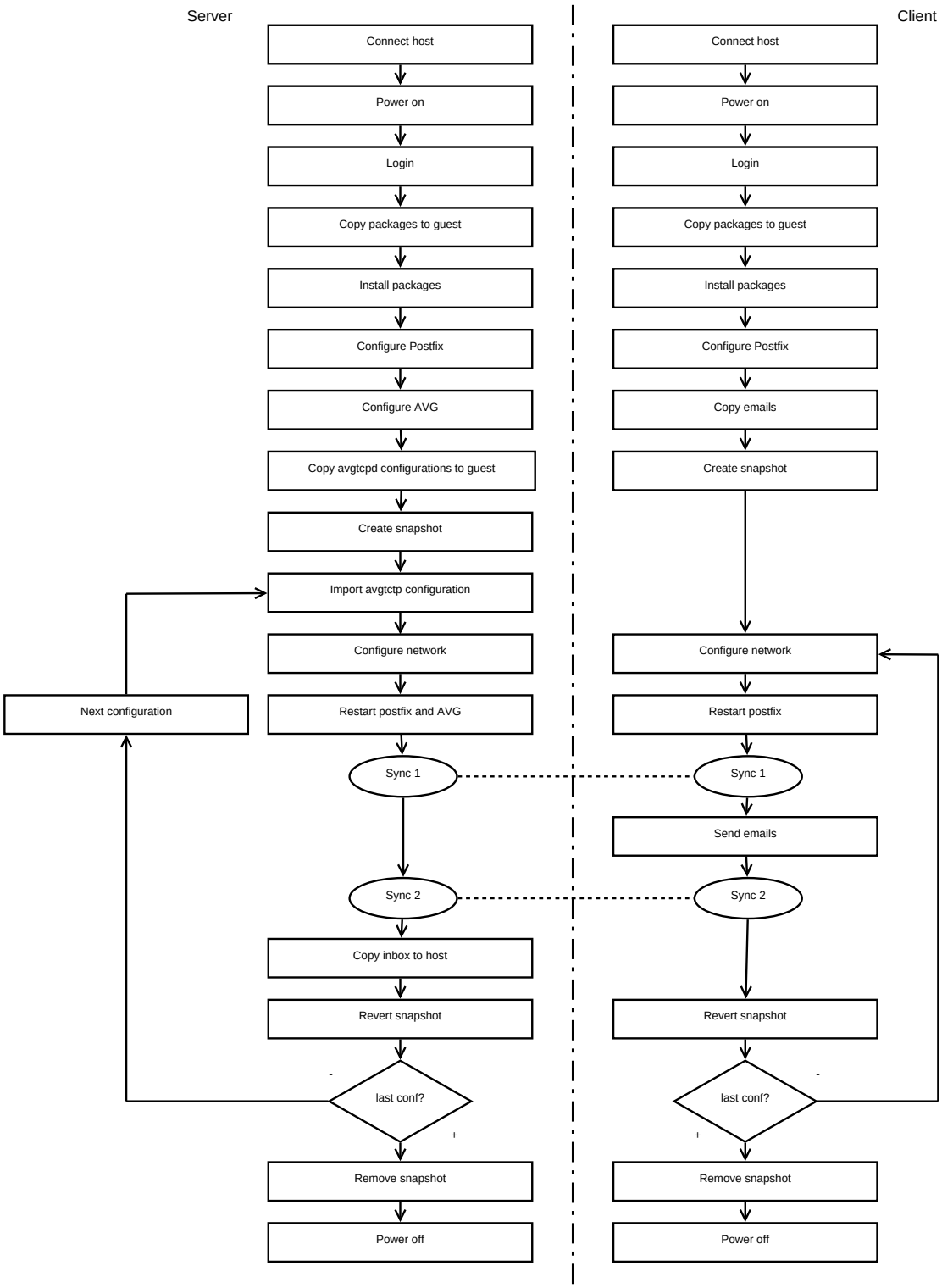
   Vývojový diagram tohoto testu je na obrázku 5.6

   Vstupy:

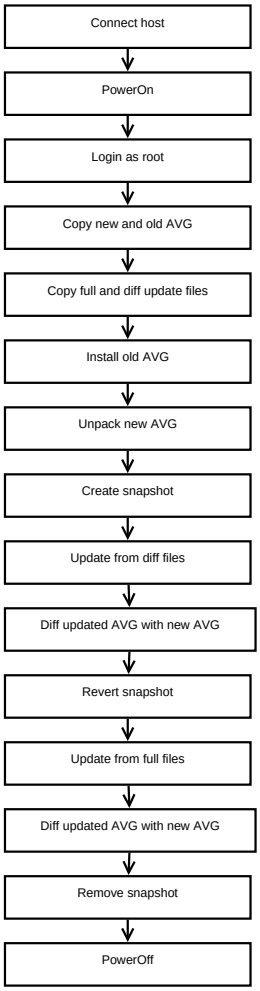
   - balíček AVG,
   - příkaz k instalaci balíčku,
   - cesta k vmx souboru virtuálního stroje,
   - cesta adresáře s aktualizacemi,
   - nový balíček AVG,
   - cesta, kam má být uložen výsledek testu.

   Výstup:

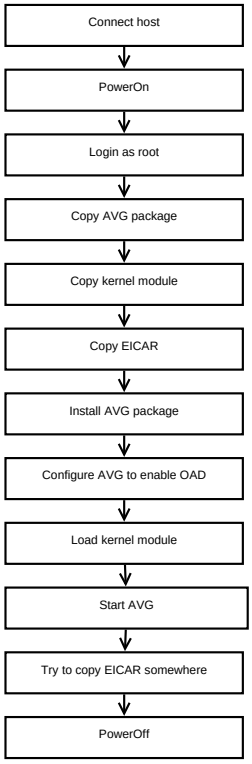
   - Soubor se seznamem rozdílů.



Obrázek 5.5: Diagram testu detekčních schopností při různých konfiguracích avgtcpd



Obrázek 5.6: Vývojový diagram testu aktualizací



Obrázek 5.7: Vývojový diagram testu OAD

7. Test avgoad – Testována je funkčnost OAD. Při testu je nahrán jeden z podporovaných modulů jádra pro předávání souborů k antivirové kontrole. Kontrolovány jsou soubory, ke kterým uživatelé přistupují. Diagram popisující tento test je na obrázku 5.7.

   Vstupy:

   - balíček AVG,
   - příkaz k instalaci balíčku,
   - cesta k vmx souboru virtuálního stroje,
   - cesta k modulu jádra daného OS,
   - cesta, kam má být uložen výsledek testu.

   Výstup:

   - Soubor s výsledky testů.

8. Test avgvvctl – Tento test je rozšířením předešlého testu. K infikovanému souboru je přistupováno několikrát s různým nastavením OAD. Kontrolován je obsah VV. Vývojový diagram je na obrázku 5.8.

   Vstupy:

   - balíček AVG,
   - příkaz k instalaci balíčku,
   - cesta k vmx souboru virtuálního stroje,
   - cesta k modulu jádra daného OS,
   - cesta, kam má být uložen výsledek testu.

   Výstup:

   - Soubory s obsahem VV při různém nastavení.

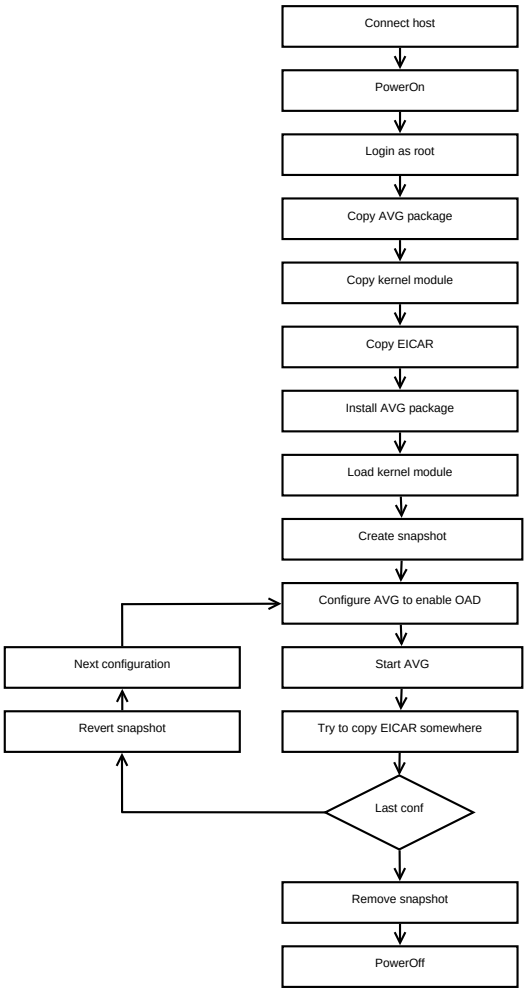
9. Test avgspmctl – Test prověřuje, že se po spuštění učení nového spamového e-mailu změní antispamová databáze. Grafický popis poskytuje diagram 5.10.

   Vstupy:

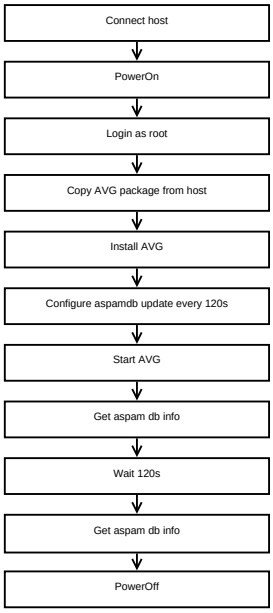
   - cesta k balíčku s AVG,
   - instalační příkaz,
   - soubor .vmx virtuálního stroje, na kterém má test probíhat,
   - cesta, kam má být uložen výsledný soubor.

   Výstup:

   - Soubor s výsledkem testu.



Obrázek 5.8: Vývojový diagram testu avg-vvctl



Obrázek 5.9: Vývojový diagram testu avgsched

10. Test avgsched – Úkolem tohoto testu je prověřit základní funkčnost plánovače. Test nastaví plánovanou aktualizaci antispamové databáze a ověří, že aktualizace byla spuštěna. Grafický návrh testu je na obrázku 5.9.

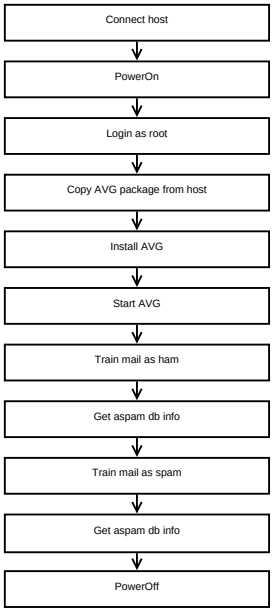
    Vstupy:

    - balíček s AVG,
    - příkaz, pomocí kterého je možné balíček s AVG nainstalovat,
    - soubor .vmx virtuálního stroje, na kterém má test probíhat,
    - cesta, kam má být uložen výsledný soubor.

    Výstup:

    - Soubor, který obsahuje výsledek testu.

Tato sada testů by měla pomoci odstranit neefektivní předávání sestavených produktů mezi odděleními AVG Technologies CZ, s. r. o. a pomoci zlepšit kvalitu výsledného produktu.



Obrázek 5.10: Vývojový diagram testu avgspmctl

Pomocí této sady byly detekovány chyby v aktuálně vyvíjené verzi produktu AVG 2011. Konkrétně se jednalo o:

- návratové hodnoty avgscan – při skenování některých druhů souborů avgscan nevrací hodnoty specifikované v manuálových stránkách;
- podmíněný sken – při nastavení detekce hrozeb pomocí parametrů příkazové řádky avgscand nedetekuje správně některé druhy hrozeb;
- odinstalace – neprobíhá správně odinstalace AVG, protože nedojde ke korektnímu ukončení všech komponent;
- aktualizace – neprobíhá správně rozdílová aktualizace;
- tcpd – nedetekuje některé infekce.

Pro zhodnocení výhodnosti vývoje a implementace těchto testů byla použita metodika Return Of Investment (ROI) z knihy [9]. ROI analýza je přiložena v příloze C.

Kapitola 6

Závěr

Úkolem práce bylo nastudovat a vylepšit překládací a sestavovací systém produktu AVG for Linux/FreeBSD. Původní systém vyžadoval při sestavování manuální kroky a neumožňoval paralelizaci. Nový systém je plně automatický a odstraňuje zbytečně prováděné kroky, díky čemuž je výrazně rychlejší (přibližně o 35 minut při celkové délce trvání překladu původního systému 66 minut). Dále nový systém umožňuje paralelní běh, což jej činí čtyřikrát rychlejším v nejhorším možném případě oproti původnímu systému, při ignoraci nutných ručních kroků. Pokud zahrneme do měření odhad délky trvání ručních kroků, je nový systém skoro desetkrát rychlejší. Přínosem automatizace je menší náchylnost na chyby obsluhy.

Dalším úkolem práce bylo také nastudovat balíčkovací systémy a vybrat vhodné balíčky pro distribuci AVG for Linux/FreeBSD. Po nastudování a zhodnocení rozšířenosti balíčkovacích systémů byl stávající výběr instalátorů (balíčky DEB a RPM a archivy TGZ a SH pro GNU/Linux a archiv TGZ pro FreeBSD) shledán vhodným. Při podrobnější analýze balíčků bylo doporučeno několik vylepšení. Tato vylepšení se týkala především opravení nedostatků nevyhovujících definicím balíčků a zavedení podepisování balíčků. Některá z těchto doporučení byla implementována a objeví se v nové verzi produktu AVG for Linux/FreeBSD.

Dále se práce zabývala automatickým testováním produktu AVG for Linux/FreeBSD. Protože se nepovedlo nalézt vhodný systém, který by bylo možné využít se stávajícím SW vybavením (VMware Workstation) používaným ve firmě AVG a zároveň by vyhovoval dalším požadavkům, byl navržen nový nástroj, který tento úkol zastane. Tento nástroj je samozřejmě možné použít i pro testování jiných produktů než je AVG. Nástroj je multiplatformní a umožňuje konfigurovat a řídit počítačovou síť tvořenou virtuálními stroji VMware.

V konečné fázi byla navržena a implementována sada automatických testů, které jsou určeny pro testování balíčků AVG for Linux/FreeBSD. Díky této sadě bylo odhaleno několik chyb v aktuálně vyvíjené verzi 2011. Součástí zadání bylo navrhnout testy pro některé komponenty, ale během návrhu se ukázalo vhodné prověřit základní funkcionalitu všech komponent. Z tohoto důvodu byl navržen alespoň jeden test pro každou z komponent. Při analýze ROI se projevilo, že implementace takto triviálních, neustále se opakujících, testů je finančně výhodná, a to i navzdory faktu, že do výdajů byly zahrnuty i náklady na implementaci speciálního nástroje.

Do budoucna by bylo možné sadu balíčků AVG for FreeBSD obohatit o balíček TGZ, který je na tomto OS používán jako jeden z hlavních zdrojů balíčků.

Vylepšen by mohl být také nástroj pro testování, a to především přidáním nových příkazů. Dále by bylo vhodné doplnit nástroj o grafické uživatelské rozhraní. Toto rozhraní nebylo ve stávající verzi implementováno, protože hlavní použití stávající verze je testo-

vání na sestavovacím serveru, ideálně automaticky, bez zásahů obsluhy, pouze s pozdějším vyhodnocením a schválením balíčků pro další testování.

Sada testů by mohla být obohacena o další testy, např. o testy výkonu, zátěžové testy a testy spolehlivosti.

Literatura

[1] BAILEY, E. C.: *Maximum RPM*. 201 West 103rd Street Indianapolis, Indiana 46290: Sams, 2000, ISBN 0672311055, 450 s.

[2] CHUNG, T.: How to sign your custom RPM package with GPG Key. `http://fedoranews.org/tchung/gpg/`, 2005-01-06 [cit. 2010-09-23].

[3] GRAHAM, D.; VAN VEENENDAAL, E.; EVANS, I.; aj.: *Foundations of Software Testing*. Thomson Learning, 2007, ISBN 1844803554.

[4] KEMP, S.: Setting up your own APT repository with upload support [online]. `http://www.debian-administration.org/articles/286`, 2005-01-05 [cit. 2010-09-23].

[5] LUCAS, M.: *Síťový operační systém FreeBSD*. Computer press, 2003, ISBN 8072267957.

[6] Manuálové stránky: Alien. `http://linux.die.net/man/1/alien`, [cit. 2011-01-08].

[7] Manuálové stránky: AVG 2011 Beta for Linux/FreeBSD. [cit. 2011-01-08].

[8] Sansing, J.: Building a Debian GNU/Linux package. `http://realeyes-tech.blogspot.com/2008/08/building-debian-gnulinux-package.html`, 2008-08-30 [cit. 2010-09-23].

[9] SUSTIN, E.; GARRETT, T.; GAUF, B.: *Implementing Automated Softrare Testing*. Addison-Wesley Professional, 2009, ISBN 0321580516.

[10] WWW stránky: The Debian GNU/Linux FAQ. `http://www.debian.org/doc/FAQ/ch-pkg_basics.en.html`, 2008-06-26 [cit. 2010-09-23].

[11] WWW stránky: Howto create a Debian repository. `http://www.rigacci.org/wiki/doku.php/doc/appunti/linux/sa/debian_repository`, 2010-04-07 [cit. 2010-09-23].

[12] WWW stránky: Hodnocení návštěvnosti za posledních 12 měsíců. `http://distrowatch.com/index.php?dataspan=52`, 2010-06-15 [cit. 2010-10-26].

[13] WWW stránky: Výsledky ankety o nejoblíbenější distribuci 2010. `http://www.abclinuxu.cz/clanky/vysledky-ankety-o-nejoblibenejsi-distribuci-2010`, 2010-06-15 [cit. 2010-10-26].

[14] WWW stránky: Linux Counter Machine Report.
http://counter.li.org/reports/machines.php, 2010-10-26 [cit. 2010-10-26].

[15] WWW stránky: AVG Rescue CD GNU/Linux. https://share.avg.com/arl/,
[cit. 2011-01-04].

[16] WWW stránky: Web AVG.
http://www.avg.com/cz-cs/avg-linux-email-server-edition, [cit. 2011-01-04].

[17] WWW stránky: GNU Make [online]. http://www.gnu.org/software/make/,
[cit. 2011-01-09].

[18] WWW stránky: Automake, Autoconf a Libtool Nástroje pro konfiguraci programů.
http://atrey.karlin.mff.cuni.cz/ 0rfelyus/aal-referat/index.html,
[cit. 2011-04-14].

[19] WWW stránky: Autotools tutorial.
http://www.lrde.epita.fr/ adl/autotools.html, [cit. 2011-04-14].

[20] WWW stránky: The Apache Ant project.
http://ant.apache.org/faq.html#history, [cit. 2011-04-15].

[21] WWW stránky: CMake. http://www.cmake.org/cmake/project/about.html,
[cit. 2011-04-15].

[22] WWW stránky: The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008.
http://pubs.opengroup.org/onlinepubs/9699919799/utilities/make.html,
[cit. 2011-04-15].

[23] WWW stránky: GNU/Linux Distribution Timeline. http://futurist.se/gldt/,
[cit. 2011-04-17].

[24] WWW stránky: VMware VIX API.
http://www.vmware.com/support/developer/vix-api/, [cit. 2011-05-10].

[25] WWW stránky: VMware Workstation.
http://www.vmware.com/products/workstation/, [cit. 2011-05-10].

[26] WWW stránky: Makeself. http://www.megastep.org/makeself/, [cit. 2011-15-04].

[27] WWW stránky: Návrhový vzor singleton.
http://objekty.vse.cz/Objekty/Vzory-Singleton, [cit. 2011-15-04].

[28] WWW stránky: Stránky projektu Autoconf.
http://www.gnu.org/software/autoconf, [cit. 2011-15-04].

[29] WWW stránky: Stránky projektu Automake.
http://www.gnu.org/software/automake/, [cit. 2011-15-04].

[30] WWW stránky: Stránky projektu Buildroot. http://buildroot.uclibc.org/,
[cit. 2011-15-04].

[31] WWW stránky: Stránky projektu Dazuko. http://www.dazuko.org/,
[cit. 2011-15-04].

[32] WWW stránky: Stránky projektu Doxygen. http://www.doxygen.org, [cit. 2011-15-04].

[33] WWW stránky: Stránky projektu GCC. http://gcc.gnu.org, [cit. 2011-15-04].

[34] WWW stránky: Stránky projektu GnuPG. http://www.gnupg.org/, [cit. 2011-15-04].

[35] WWW stránky: Stránky projektu Nokia Qt. http://qt.nokia.com/products/, [cit. 2011-15-04].

[36] WWW stránky: Stránky projektu RedirFS. http://www.redirfs.org, [cit. 2011-15-04].

[37] WWW stránky: Stránky projektu STAF/STAX. http://staf.sourceforge.net/, [cit. 2011-15-04].

[38] WWW stránky: Stránky projektu VirtualBox. http://www.virtualbox.org/, [cit. 2011-15-04].

[39] WWW stránky: Virus found in Vietnamese language pack. https://bugzilla.mozilla.org/show_bug.cgi?id=432406, [cit. 2011-15-05].

Příloha A

Rychlost překladu a sestavení AVG

Úkolem měření je porovnat rychlost překladu AVG 8.5 a AVG 2011. Překlad a sestavení bylo prováděno pomocí programu GNU/Make, *gcc-4.3* na počítači s HW konfigurací:

- Procesor: Intel Xeon W3520 2.67GHz, 4 jádra a hyper-threading;
- RAM: 4GB.

Cílem překladu je sestavený produkt se základní konfigurací ve stavu, v jakém jej uživatel nainstaluje. Při překladu nebyly generovány soubory pro aktualizace. V případě překladu AVG 8.5 nebyly provedeny žádné ruční kroky. AVG 2011 bylo překládáno s různými hodnotami parametru jobs (-j), který slouží k volbě počtu paralelně prováděných kroků překladu. Měření bylo 30 krát opakováno a byl vypočítán aritmetický průměr, rozptyl a směrodatná odchylka.

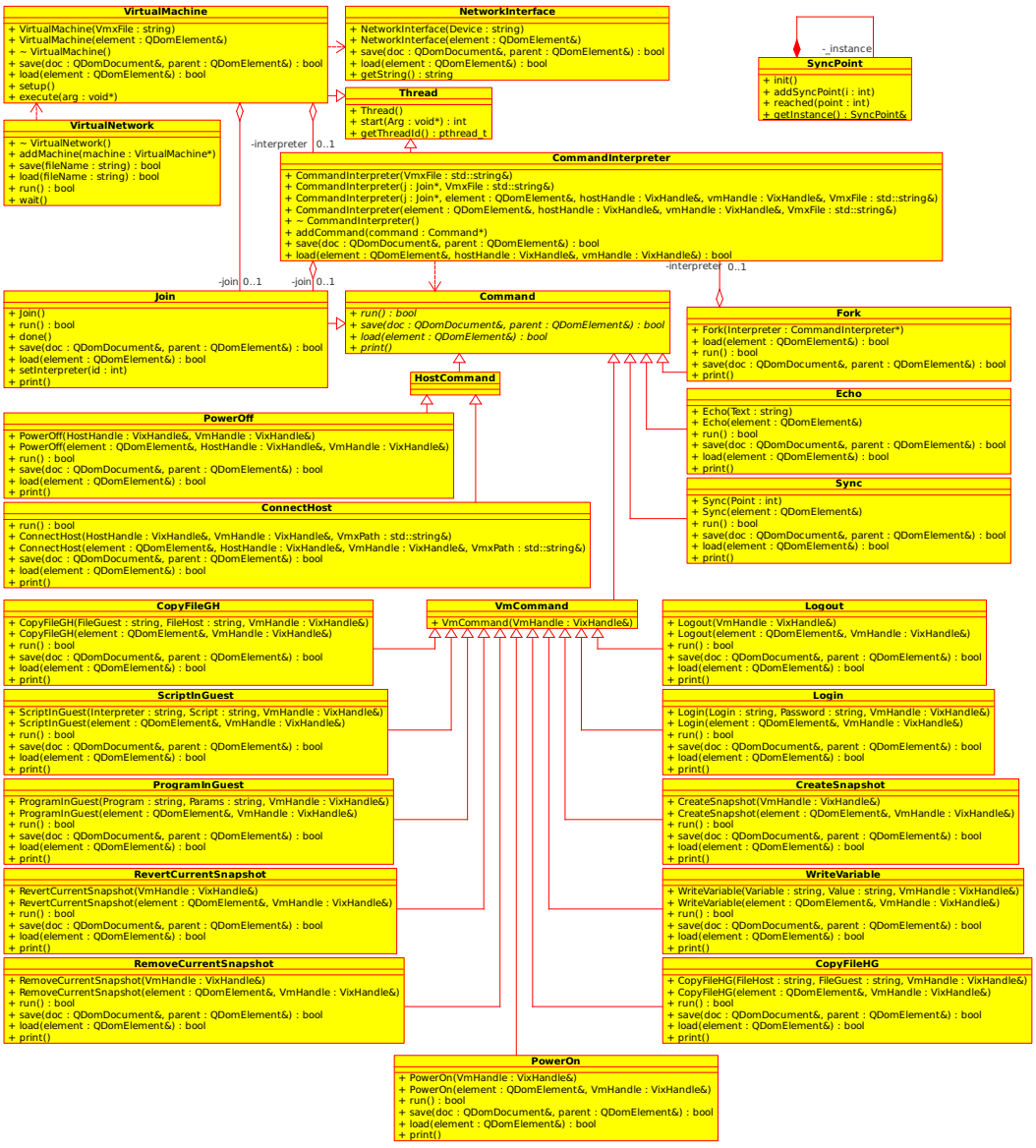
Výsledky měření jsou v tabulce A.1 a hodnoceny jsou v části 5.1.2. Skript, který byl použit pro překlad je přiložen na DVD ve složce `scripts`.

	AVG 8.5 1 vlákno [s]	AVG 2011 1 vlákno [s]	AVG 2011 4 vlákna [s]	AVG 2011 8 vláken [s]
	1569, 15	1686, 53	468, 33	397, 05
	1547, 01	1638, 05	468, 81	399, 75
	1551, 46	1650, 63	472, 09	396, 3
	1558, 87	1641, 13	468, 98	395, 18
	1551	1637, 91	468, 35	395, 51
	1555, 16	1637, 1	470, 43	398, 92
	1552, 96	1641, 25	468, 91	400, 72
	1556, 93	1636, 85	470, 19	400, 07
	1548, 93	1642, 2	467, 91	398, 44
	1547, 21	1638, 88	473, 06	395, 03
	1564, 42	1668, 26	472, 64	402, 2
	1547, 89	1647, 94	473, 98	403, 24
	1542, 08	1639, 45	472, 49	402, 2
	1545, 01	1639, 22	472, 9	399, 31
	1555, 2	1638, 4	472, 35	399, 96
	1550, 17	1640, 67	471, 28	399, 49
	1547, 88	1638, 81	472, 21	400, 13
	1548, 88	1637, 91	471, 15	400, 66
	1541, 86	1657, 27	474, 33	397, 43
	1547, 7	1649, 25	476, 05	403, 87
	1558, 94	1687, 64	495, 03	403, 03
	1545, 82	1689, 73	479, 87	402, 43
	1543, 88	1682, 94	481, 69	406, 04
	1545, 88	1684, 04	486, 24	402, 75
	1542, 21	1665, 03	480, 79	403, 18
	1544, 33	1665, 43	486, 11	398, 52
	1544, 28	1656, 13	481, 74	402, 43
	1547, 99	1649, 8	480, 5	404, 88
	1543, 51	1650, 3	489, 7	403, 57
	1548, 88	1648, 07	481, 16	404, 64
	1551, 39	1670, 66	504, 25	405, 47
	1547, 03	1672, 41	491, 02	407, 96
	1547, 54	1667, 03	478, 28	407, 62
	1550, 23	1662, 18	502, 97	405, 49
	1547, 1	1653, 97	484, 73	405, 13
	1551, 12	1652, 69	479, 81	406, 25
	1547, 58	1654, 68	486, 54	409, 67
	1549, 53	1661, 23	485, 19	407, 41
	1550, 3	1655, 87	483, 86	407, 56
	1555, 13	1655, 13	488, 49	408, 9
Aritmetický průměr	1549, 81	1654, 82	478, 86	402, 21
Rozptyl	34, 27	252, 05	86, 57	15, 63
Směrodatná odchylka	5, 85	15, 88	9, 3	3, 95

Tabulka A.1: Tabulka rychlosti překladu AVG 8.5 a AVG 2011

Příloha B

Diagram tříd



Obrázek B.1: Diagram tříd testovacího nástroje

Příloha C

Return Of Investment

Overall Test Automation Savings Worksheet				
Tasks	**Automated Testing Time Saving [hours]**		**Tasks**	**Automated Testing Time Saving [Kč]**
Test setup time saving	0,00		Test setup time savings	0,00
Test development time savings	-16,67		Test development cost savings	-3333,33
Test execution time saving	475,00		Test execution cost saving	76000,00
Test evaluation/diagnostics time saving	112,50		Test evaluation/diagnostics cost saving	18000,00
Test tool development	-160,00		Other automation costs	-32000,00
Hours	**410,83**		Sum	**58666,67**
Days [8-hours days]	51,35			
Months (20-day months)	**2,57**			

Tabulka C.1: Overall Test Automation Saving Worksheet

Teting Phase:	**Test Environment Setup Time Saving Worksheet**			
	Manual Test Setup		**Automated Test Setup**	
Basic tests	Test setup time [minutes]		Test setup time [minutes]	
	Other		Other	
	Number of iterations		Number of iterations	
	Total test setup time [hours]		Total test setup time [hours]	
	Test setup time savings [hours]			

Tabulka C.2: Test Environment Setup Time Saving Worksheet

Test Development Time Saving Worksheet			
Manual Test Development		**Automated Test Development**	
Number of tests planned	20,00	Number of tests planned	20,00
Time to develop 1 test [minutes]	10,00	Time to develop 1 test [minutes]	60,00
Total test development time [hours]	3,33	Total test development time [hours]	20,00
Test development time saving [hours]	-16,67		

Tabulka C.3: Test Developement Time Saving Worksheet

Test Execution Time Saving Worksheet			
Manual Test Execution		**Automated Test Execution**	
Number of tests planned	20,00	Number of tests planned	20,00
Estimation time to execute each test [minutes]	20,00	Estimation time to execute each test [minutes] (manual/20)	1,00
Estiomated number of test iterations	75,00	Estiomated number of test iterations	75,00
Total test estimate to execute manual test [hours]	500,00	Total test estimate to execute automated test [hours]	25,00
Total time saving [hours]	475,00		

Tabulka C.4: Test Execution Time Saving Worksheet

Test Evaluation/Diagnostics Time Saving Worksheet			
Manual Test Evaluation/Diagnostics		**Automated Test Evaluation/Diagnostics**	
Number of test outputs to diagnose	20,00	Number of test outpust to diagnose	20,00
Estimation time to evaluate/compare/diagnose test output [minutes]	5,00	Estimation time to evaluate/compare/diagnose test output [minutes] (manual/10)	0,50
Estiomated number of test iterations	75,00	Estiomated number of test iterations	75,00
Total test evaluation time [hours]	125,00	Total test evaluation time [hours]	12,50
Test evaluation/diagnostics time savings [hours]	112,50		

Tabulka C.5: Test Evaluation/Diagnostics Time Saving Worksheet

Počet spuštění testů (Estiomated number of test iterations) odpovídá počtů sestavených verzí AVG ve verzi 8.5. Počet testů (Number of tests planned) a výstupů testů (Number of test outpust to diagnose) odpovídá počtu automatizovaných testů (testy jsou rozděleny na jednotlivé části). Čas vyhodnocení testu (Estimation time to evaluate/compare/diagnose test output) a čas spuštění testu (Estimation time to execute each test) odpovídá průměrnému času vyhodnocení a spuštění této třídy testů (konzultováno s testery AVG). Tabulka C.2 nebyla pro tuto analýzu použita, protože čas potřebný k nastavení prostředí testu byl zahrnut do času spuštění.

Cílem této analýzy je vyhodnotit zda, případně jak moc, se vyplatí AVG automatizovat základní testy. Z analýzy vyplývá, že i přes náklady na vývoj testovací aplikace AVG ušetří přibližně 411 hodin pracovního času, což odpovídá přibližně 58667Kč.

Příloha D

Obsah přiloženého DVD

/	DVD v příloze
scripts	skripty použité při tvorbě práce
test_files	soubory, které je možné zadat jako parametry různým testům
avg2011	build 1315 AVG 2011 beta
avg85	build 874 AVG 8.5 for Linux
mails	testovací sada emailů
oad_modules	modul jádra OAD
tcpd_configs	různé nastavení detekce avgtcpd
tester	zdrojové kódy testovacího nástroje
tests	skripty jednotlivých testů
avgscan	pevně dané soubory testu avgscan
postfix	pevně dané soubory testu tcpd
smoke	pevně dané soubory smoke testu
text	text této práce
vmware	ukázkové obrazy Vmware s Debian 6