Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Nalezení aktualizacemi ovlivněných aplikací

2015

Jakub Kadlčík

Vedoucí práce: doc. RNDr. Michal Krupka, Ph.D.

Studijní obor: Informatika, prezenční forma

Bibliografické údaje

Autor:	Jakub Kadlčík
Název práce:	Nalezení aktualizacemi ovlivněných aplikací
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2015
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	doc. RNDr. Michal Krupka, Ph.D.
Počet stran:	44
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Jakub Kadlčík
Title:	Determine applications affected by upgrade
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2015
Study field:	Computer Science, full-time form
Supervisor:	doc. RNDr. Michal Krupka, Ph.D.
Page count:	44
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce se zabývá problematikou „aktualizacemi ovlivněných aplikací“, konkrétně pak jejich nalezením a doporučením nejlepšího způsobu, jak docílit jejich restartování. V teoretické části řeší otázkou důležitosti tohoto problému a pojednává o možných algoritmech. Součástí práce je vytvoření nástroje pro vyhledávání ovlivněných aplikací na linuxové distribuci Fedora. Tento nástroj by mělo být možné použít také skrze rozšíření balíčkovacího systému DNF.

Synopsis

This thesis deals with the topic of „applications affected by upgrade“, particularly of their determination and recommending the best way of achieving their restart. In the theoretical part it shows the relevance of the problem and describes possible algorithms. This thesis also includes the tool for determining affected applications on Fedora linux distribution which should be possible to use via an extension of the DNF package manager.

Klíčová slova: aktualizace systému, ovlivněné aplikace, Python, Fedora, DNF

Keywords: system updates, affected applications, Python, Fedora, DNF

Rád bych poděkoval doc. RNDr. Michalu Krupkovi, Ph.D. za cenné rady a pomoc při tvorbě této práce. Děkuji také společnosti Red Hat, Inc., jmenovitě pak Ing. Miroslavu Suchému, za inspirativní spolupráci.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Ve většině distribucí GNU/Linuxu se software standardně spravuje prostřednictvím balíčkovacího systému. Jedná se o nástroj zajišťující evidenci nainstalovaného a dostupného softwaru, jeho instalaci a následnou aktualizaci. Nové distribuce většinou vznikají odvozením jiné, již existující distribuce a v tomto případě bývá zachován typ balíčků i balíčkovací systém. Můžeme tedy říct, že distribucí je mnoho, ale balíčkovaích systémů se používá jen velmi málo. Konkrétně lze hovořit o distribucích GNU/Linuxu používající balíčky RPM[1], DEB[2] a nebo jiný typ balíčku. V posledním případě jsou balíčky popsány případ od případu velmi specifickým předpisem. V rámci této práce se budeme zabývat linuxovou distribucí Fedora jakožto reprezentantem první skupiny.

Fedora vznikla jako nekomerční odnož Red Hat Enterprise Linuxu a zachovala stávající systém balíčků RPM. Od své první verze využívá nástavbu nad RPM zvanou YUM. Ten se po čase stal z mnoha technickůch důvodů nedostatečný a proto se objevila alternativa pojmenovaná DNF. V současné době lze paralelně využívat oba tyto nástroje, avšak je snaha YUM kompletně nahradit za pomocí DNF.

Když spustíme aplikaci, do paměti se načtou knihovny a soubory potřebné k jejímu běhu. Pokud je některá z knihoven, nebo samotná aplikace po dobu jejího běhu aktualizována, v paměti stále zůstanou původní soubory, přestože fyzicky už nemusí existovat (mohou být smazány, nebo nahrazeny novějšími verzemi). Zjistit, které, v systému spuštěné, aplikace využívají takto neaktuální soubory, není pro uživatele triviální. V současné době sice existují nástroje, které tento problém řeší, avšak jsou z mnoha důvodů nedostatečné, viz kapitola 8.

Na první pohled by se mohlo zdát, že postrádá smysl, zabývat se takovým problémem, protože jeho vyřešení nepřinese nic zajímavého. Ve skutečnosti však mohou být důsledky velmi nebezpečné. Pokud například aktualizace opravuje bezpečnostní chybu staré verze, běžící aplikace zůstanou neopravené. K jejich opravě dojde až jejich restartováním. Uživateli se tak může stát, že nainstaluje záplatu, bude si myslet, že jeho systém je opět bezpečný a přitom stále běží několik nebezpečných procesů. Další problém se týká spíše uživatelů *unstable* distribucí, kterým do systému mohou přicházet zpětně nekompatibilní balíčky. Potom se může stát, že některá služba odmítne nastartovat, protože nová verze nepovoluje stávající konfiguraci, nebo vlivem nepřesně specifikovaných závislostí zůstane v systému nekompatibilní verze knihovny. Takovýchto případů bychom si rádi všimli, co nejdříve to bude možné, nikoliv až v kritické chvíli, kdy například dojde k restartování serveru.

[1] RPM = Red Hat Package Manager
[2] DEB = Debian software package

Práce si klade za cíl vytvořit software, který bude řešit problém *nalezení aktualizacemi ovlivněných aplikací*. Software by měl najít všechny programy využívající neaktuální soubory a doporučit jejich restartování. Pro různé typy programů by měla být vypsána jiná doporučení a nápověda, jakým způsobem to lze provést.

2 Požadavky

Tato práce vznikala ve spolupráci se společností Red Hat, Inc., přičemž následující požadavky byly zhotoveny na základě již existujícího zadání.

Teoretická část práce by se měla zabývat problematikou aktualizacemi ovlivněných aplikací a otázkou jejich vyhledání. Budou uvedeny a vysvětleny algoritmy pro řešení tohoto problému. Výsledkem práce bude software implementující některý z teoreticky popsaných algoritmů.

Software bude implementován v programovacím jazyce Python a jeho cílovou platformou bude linuxová distribuce Fedora, pro kterou bude k dispozici instalační balíček. Vývoj software bude probíhat pod svobodnou licencí splňující kritéria pro přidání balíčku do oficiálních repozitářů této distribuce. Zdrojový kód programu bude umístěný na libovolném veřejném úložišti a bude komukoliv k dispozici.
Software by měl disponovat textovým uživatelským rozhraním a mělo by být možné jej používat přímo skrze balíčkovací systém. Z tohoto důvodu bude taktéž popsána problematika tvorby rozšíření pro balíčkovací systém DNF.

3 Analýza problému

V této kapitole rozebereme problematiku na dílčí části, navrhneme algoritmy pro jejich vyřešení a provedeme úvahu nad vhodností jejich použití. Nejdříve se zaměříme na samotné získání aktualizacemi ovlivněných aplikací, poté zvážíme univerzálnost navrhovaného řešení napříč operačními systémy a nakonec se podíváme na to, jakým způsobem lze nalezené aplikace restartovat.

3.1 Seznam ovlivněných aplikací – Ilustrativně

Hlavním účelem programu bude nalezení aktualizacemi ovlivněných, běžících aplikací. Řešení se zdá být velmi přímočaré. Nejdříve zjistit seznam balíčků, které byly od spuštění systému modifikovány. Následně pro každý balíček zjistit seznam souborů, které poskytuje. Poté zjistit, které procesy využívají některý z množiny modifikovaných souborů. Tedy pro každý spuštěný proces zjistit seznam souborů jež využívá a najít shodu se soubory poskytovanými modifikovanými balíčky.

```
packages = seznam balíčků modifikovaných od spuštění systému
processes = seznam všech spuštěných procesů
for package in packages:
    for process in processes:
        if množinový_průnik(package_files, process_files) != prázdná množina
            print process
```

Zdrojový kód 1: Algoritmus pro vypsání seznamu ovlivněných aplikací

Algoritmus na první pohled vypadá velmi jednoduše a zřejmě vypíše korektní množinu procesů. Mohlo by se tedy zdát, že máme hotovo, nicméně jsme nevzali v úvahu časovou složitost takového algoritmu. Ta, jak si ukážeme níže, je bohužel velmi nepříznivá, což by vedlo k útrpně pomalému vyhledávání.

Časová složitost

Časovou složitost lze vyjádřit následovně. Průchod přes všechny balíky je lineární, tedy $O(n)$. Pro každý balík se prochází všemi procesy, takže násobíme $*O(m)$. Protože jsou obě m i n spočetně velké a nejsou to konstanty, místo $O(n * m)$ vyjádříme $O(n^2)$.

Dále počítáme složitost množinového průniku. V případě jeho naivní implementace, která porovnává každý prvek jedné množiny s každým prvkem druhé, dostaneme kvadratickou složitost $O(n^2)$. V případě optimalizovanějšího návrhu si polepšíme na $O(n)$.

Množinový průnik je zjišťován pro každou kombinaci balíčku a procesu, proto původní $O(n^2)$ násobíme $O(n)$ (případně $O(n^2)$ při naivní implementaci). Výslednou časovou složitost ukázaného algoritmu tedy vyčíslíme jako $O(n^3)$, případně $O(n^4)$.

3.2 Seznam ovlivněných aplikací – Vylepšení

Klíčem k rychlejšímu získání chtěných procesů je zvolení vhodné vyhledávací struktury pro data. Vezměme si fakt, že mnoho procesů využívá stejné soubory (např. knihovny). Naivní algoritmus tedy jeden konkrétní soubor musí ověřit hned několikrát – pro každý proces, který jej využívá.

Tento nedostatek můžeme řešit tak, že nebudeme zkoumat proces po procesu, ale místo toho vezmeme množinu všech souborů v paměti, přičemž ke každému souboru si poznačíme procesy, které jej využívají. Pro nejrychlejší možné vyhledávání v množině souborů, zvolíme strom typu BTree jako strukturu pro uložení dat. Zaplatíme sice zvýšenou spotřebou paměti, nicméně rychlostní zlepšení bude na první pohled zjevné.

```
packages = seznam balíčků modifikovaných od spuštění systému
processes_files = BTree(...)
for package in packages:
    for package_file in package_files:
        processes = processes_files[package_file]
        if processes != prázdná množina:
            print processes
```

Zdrojový kód 2: Vylepšený algoritmus pro vypsání seznamu ovlivněných aplikací

Časová složitost

Při vyhledávání je pro každý balíček procházen seznam jím poskytovaných souborů. Složitost tohoto problému je analogická předchozí situaci, tedy $O(n^2)$. Co se ovšem změnilo, je problém nalezení všech procesů, využívajících daný soubor. Nyní si stačí vyzvednout jejich seznam jednorázovým přístupem do stromové struktury typu BTree. Složitost tohoto problému je $O(log(n))$. Poskládáním jednotlivých kousků problému dohromady, získáváme časovou složitost celého algoritmu vyjádřenou jako $O(n^2 * log(n))$.

3.3 Informace o aplikaci

Na chvíli se zamysleme nad přenositelností nástroje mezi různými linuxovými distribucemi. Položme si následující otázku, je možné zjišťovat informace o aplikacích stejným způsobem na libovolném linuxovém systému? Prozradím, že část z nich skutečně ano, avšak ne všechny.

Rozlišujeme totiž dva druhy informací. Pracujeme se spuštěnými aplikacemi, tedy lze hovořit o procesech spuštěných v systému. Každý proces má nějaký PID[3], spustil jej nějaký uživatel, běží určitou dobu, apod.

Dále lze uvažovat, že každá[4] aplikace je poskytována distribučním balíčkem. Každý takový balíček poskytuje informace o tom, k čemu slouží, v jaké je verzi, jak dlouho je nainstalován, apod.

Lze předpokládat, že informace o procesech budou získatelné jednotným způsobem napříč celým spektrem linuxových distribucí, avšak informace o balíčku budou závislé na jeho typu. Jejich struktura se totiž výrazně liší, používají jiné názvy atributů a tak podobně.

3.4 Jak aplikaci restartovat?

Aplikace můžeme rozdělit do několika kategorií. Zřejmě budou existovat takové, jež nemůžeme restartovat jinak, než rebootováním celého operačního systému. Příkladem budiž init, či jaderné procesy. Dále jistě budou v systémech s grafickým uživatelským rozhraním aplikace, k jejichž restartování bude potřeba řestartovat celé grafické prostředí. Příkladem mohou být správci oken a aplikace související

[3] PID (Process ID) je unikátní atribut pro identifikaci procesu

[4] Každá standardně nainstalovaná

s uživatelským sezením[5]. V těchto případech nezbývá jiná možnost, než provést požadovanou operaci jako reboot počítače, či odhlášení a opětovné přihlášení uživatele do grafického režimu.

Zajímavější situace je v případě aplikací, které lze restartovat pomocí příkazu v příkazové řádce. Ta je typická, nikoliv však výhradní, pro tzv. daemony, či služby. Napříč velkým spektrem distribucí a správců služeb, lze k jejich restartování standardně použít příkaz `service` *název_služby* `restart`. Podobně lze tímto způsobem přistupovat i ke spoustě běžných aplikací, u nich se ale budou jednotlivé příkazy fundamentálně lišit.

Všechny ostatní aplikace spadají do poslední skupiny. Uživatel je musí ručně ukončit a znovu spustit. Nezáleží na tom, zda jsou grafické a bude tak učiněno křížkem v záhlaví okna, nebo spuštěné v příkazové řádce a budou ukončeny jiným způsobem. Tato poslední kategorie není bezpečně automatizovatelná a proto musí zůstat v plné režii uživatele.

4 Linuxová distribuce Fedora

Fedora je linuxová distribuce GNU/Linuxu vzniklá jako nekomerční odnož Red Hat Enterprise Linuxu. Odtud převzala stávající systém balíčků RPM, nad kterým poskytuje nástavby v podobě správců balíčků YUM a novějším DNF. Fedoru vyvíjí komunita vývojářů okolo Fedora Project, sponzorovaného firmou Red Hat. Projekt vznikl v roce 2003, kdy se firma rozhodla rozdělit Red Hat Linux na komerční Red Hat Enterprise Linux a komunitní Fedoru.

Správce balíčků YUM byl nedílnou součástí Fedory již od jejího prvního vydání Fedora Core 1, avšak v aktuálně stabilní verzi Fedora 22 došlo k jeho nahrazení nástrojem DNF [1]. Nyní lze používat oba nástroje paralelně, avšak YUM je označen za zastaralý a v příštím vydání bude ze systému kompletně odstraněn. Důvody nahrazení jdou mimo rámec této práce, nicméně jedním z nich, pro tuto práci významným, je modularita systému DNF.

Projekt se skládá z knihovny hawkey, poskytující aplikační rozhraní pro jazyky C a Python ke knihovně libsolv, samotného nástroje DNF a dále pak sbírkami rozšíření `dnf-plugins-core` a `dnf-plugins-extras`, pomocí kterých lze vylepšovat jeho funkcionalitu. Do těchto sbírek lze za určitých podmínek nechat zahrnout i vlastní rozšíření.

[5] Známější spíše v anglickém originále jako session

5 Programovací jazyk Python

Jedním z požadavků této práce je, aby výsledný produkt byl realizován v programovacím jazyce Python. Proč je tomu tak a proč zrovna Python? Na tuto otázku existuje velmi jednoduchá odpověď. Nový balíčkovací systém distribuce Fedora, DNF, je napsán právě v tomto jazyce a poskytuje v něm API pro tvorbu rozšíření třetích stran [2]. Nehledě na to, Python disponuje několika vlastnostmi, ve kterých konkurence zaostává.

5.1 Historie

Koncept jazyka Python vznikl na konci osmdesátých let dvacátého století v Amsterdamském CWI[6]. Jeho autorem je Guido van Rossum, který dosud stále stojí v čele jeho vývoje. Při návrhu Pythonu se inspiroval jazykem ABC, na kterém v té době pracoval. Implementace započala v prosinci 1989 a první stabilní verze 1.0 vyšla o několik let později, v roce 1994. Od té doby se vývoj posunul o dvě major verze dopředu. Řada 2.x.x odstartovala v roce 2000 a je stále velmi populární obzvláště v komerční sféře. V roce 2008 vyšla verze 3.0, která měla být nástupcem dvojkové řady, avšak kvůli neúplné zpětné kompatibilitě [3] probíhá vývoj obou řad paralelně. Aktuálně lze tedy Python získat ve dvou stabilních verzích. V době psaní této práce se jedná konkrétně o verze 2.7.10 [4] a 3.4.3 [5].

5.2 Vlastnosti

Python je platformově nezávislý, multi-paradigmatický programovací jazyk, jehož výchozím stylem je strukturované, případně objektově orientované programování. Poskytuje však mnohé prvky funkcionálního paradigmatu, jako například lambda výrazy, iterátory, ne-destruktivní varianty většiny standardních funkcí, či například funkce `map`, `filter`, nebo `apply` dobře známé z ostatních funkcionálních jazyků.

Typování je v tomto jazyce dynamické, avšak silné. Kupříkladu operaci sčítání tedy nelze provést na jednom argumentu typu číslo a druhém argumentu typu řetězec, i kdyby v něm byla zapsána číselná hodnota. K žádným koercím a implicitnímu přetypování v tomto případě nedojde a výsledkem operace bude chyba. Velmi výrazným rysem je takzvaný duck typing, který preferuje provedení dané operace s objektem, aniž by se předem ověřoval jeho typ. Při selhání zjevně objekt nebyl požadovaného typu a situace se vyřeší výjimkou. V opačném případě objekt poskytoval požadované rozraní a byl tedy správného typu.

Standardní knihovna se snaží být „rozumně“ velká a všechna dodatečná funkcionalita je řešena formou rozšíření. Hlavní implementace Pythonu zvaná CPython je nap-

[6] Centrum Wiskunde & Informatica (English: National Research Institute for Mathematics and Computer Science)

saná v jazyce C standardu C89. Tento překladač proramy kompiluje do přechodného bytekódu (soubor s příponou `.pyc`), který je následně prováděn virtuálním strojem. O správu paměti se stará garbage collector.

5.3 Filozofie

Rady a doporučení, jak by měl vypadat zdrojový kód v jazyce Python, sepsal Tim Peters v dokumentu PEP 20 – The Zen of Python [6]. Jak sám píše na začátku dokumentu, principy shrnul do dvaceti aforismů, z nichž však pouze devatenáct bylo zapsáno. Následuje ukázka samotného Zenu:

> Beautiful is better than ugly.
> Explicit is better than implicit.
> Simple is better than complex.
> Complex is better than complicated.
> Flat is better than nested.
> Sparse is better than dense.
> Readability counts.

Zbývajících dvanáct veršů naleznete ve zmíněném dokumentu, případně v interpretru jazyka, pomocí příkazu `import this`.

5.4 Syntax a sémantika

Základním syntaktickým rysem, kterým se syntax liší od jazyků rodiny C je použití odsazení k vyhrazení bloku kódu namísto složených závorek. Přídává klíčové slovo `with`, které umožní otevřít zdroj, implementující metody `__enter__` a `__exit__`, pro daný blok kódu. Konstrukt se sám postará o ukončení práce se zdrojem a to i v případě, kdy ve vykonávaném kódu dojde k chybě. V jazyce nenajdeme konstrukci `switch`. Návrh o její přídání byl oficiálně zamítnut, viz PEP 3103 [7]. K dispozici máme několik základních datových struktur jako jsou seznamy (obdoba pole), slovníky (obdoba asociativního pole), tuply, či množiny. Všechny se používají velmi snadno, například seznam lze vytvořit zapsáním jeho prvků mezi hranaté závoky.

Doporučené konvence pro formátování zrojového kódu popisuje dokument PEP 8 [8]. Pojednává například o způsobu odsazování, doporučené šírce řádku, komentářích, způsobech pojmenování různých elementů a podobně.

6 Implementace

Již jsme se seznámili s prostředím linuxových systémů a problematikou hledání aktualizacemi ovlivněných aplikací. Ukázali jsme si také konkrétní vyhledávací algoritmy předpokládající jisté vstupní parametry, jako například seznam modifikovaných balíčků, nebo seznam všech procesů. Samotný algoritmus pak zjišťoval seznam souborů poskytovaných daným balíčkem, či seznam souborů využívaných daným procesem. V této kapitole si vysvětlíme, jakým způsobem lze v jazyce Python zmíněné vstupní parametry získat a také, jakým způsobem realizovat potřebné operace.

6.1 Práce s procesy

Správu procesů v jazyce Python zajišťuje modul `psutil`, který je ve Fedoře k dispozici pod názvem `python-psutil`. Tento modul umožňuje získání seznamu všech aktuálně běžících procesů a poskytuje přístup ke všem jejich atributům, které lze v linuxových systémech zjistit. Mimo to poskytuje rozhraní pro získávání informací o paměti, procesoru, pevných discích a síťových rozhraních [9].

Nás ovšem zajímá pouze práce s procesy. Ukázku, jak vypadá rozhraní objektu procesu, vidíme v následujícím příkladu.

```
>>> import psutil
>>> p = psutil.Process(pid)

>>> p.pid
13005

>>> p.name()
'gvim'

>>> p.username()
'frostyx'

>>> p.create_time()
1408897172.2

>>> p
<psutil.Process(pid=13005, name='gvim') at 3071122124>
```

Zdrojový kód 3: Základní práce s procesy v psutil

Můžeme požadovat i pokročilejší funkce, jako například zjištění souborů, které daný proces využívá. Pro následující ukázku, stejně tak jako pro tu předchozí, předpokládejme identifikátor některého z procesů uložený v proměnné `pid`.

```
import psutil
p = psutil.Process(pid)
for mmap in p.get_memory_maps():
    print mmap.path
```

Zdrojový kód 4: Výpis všech souborů, které proces využívá

Výpis tohoto ukázkového kódů může vypadat následovně:

```
/usr/lib/libxcb-dri2.so.0.0.0
/usr/lib/python2.7/lib-dynload/array.so
/usr/lib/gio/modules/libgvfsdbus.so
/usr/lib/libgio-2.0.so.0.4000.0
/home/frostyx/.cache/fontconfig/c930fb9cfe873e28f677c512e5c2fd74-le32d4.cache-4
/usr/lib/python2.7/lib-dynload/datetime.so
/usr/lib/libXt.so.6.0.0
...
```

Zdrojový kód 5: Ukázka výstupu předchozího kódu

Jak jsme mohli vidět, objekty procesů jsou vytvářeny na základě jejich identifikátoru PID. Zjištění seznamu unikátních identifikátorů je tedy klíčkovou záležitostí, kterou pro nás zajišťuje funkce `psutil.get_pid_list()`.

6.1.1 Problémy při práci s procesy

Při práci s procesy narážíme na několik problémů, které bychom neustále měli mít na paměti.

K prvnímu významnému jevu dochází, když je vytvořen objekt reprezentující proces, poté je daný proces ukončen a následovně dojde k pokusu o přístup k atributu jeho objektu. Obdobná situace nastává v případě získání seznamu identifikátorů všech spuštěných procesů a následného vytvoření objektů na jejich základě. Může se totiž stát, že došlo k ukončení některých procesů mezi časem získání jejich identifikátoru a časem vytváření objektů. V těchto případech dochází k výjimce, se kterou si však v naší situaci není potřeba příliš lámat hlavu. Proces byl již ukončen, takže pro naše účely není ničím zajímavý[7]. Je však důležité si tuto problematiku uvědomit.

Dále si lze všimnout poměrně zvláštního jevu. Tím jsou podivné řetězce na konci názvů, některých procesem používaných, souborů. Množinu všech možných přípon, které se mohou vyskytnout, nedovedu přesně určit, nicméně v průběhu tohoto projektu jsem se setkal s následujícími případy:

```
/usr/lib64/kde4/kded_networkmanagement.so;53c7cd86
/usr/lib64/firefox/plugin-container.#prelink#.N3n7Rk (deleted)
/usr/bin/gvim#new (deleted)
```

Zdrojový kód 6: Artefakty v názvech souborů

Tyto soubory samozřejmě fyzicky neexistují. Například na konci prvního zmíněného souboru přebývá řetězec `;53c7cd86`. Po jeho vyjmutí z názvu, však získáme cestu ke skutečně existujícímu souboru poskytovanému balíčkem `kde-plasma-nm`. Obdobně to platí i pro příponu `.#prelink#` a jí následovaný identifikátor, nebo příponu `#new`. Je potřeba si také dát pozor na příznak `(deleted)`, který se může objevit i nezávisle na těchto příponách.

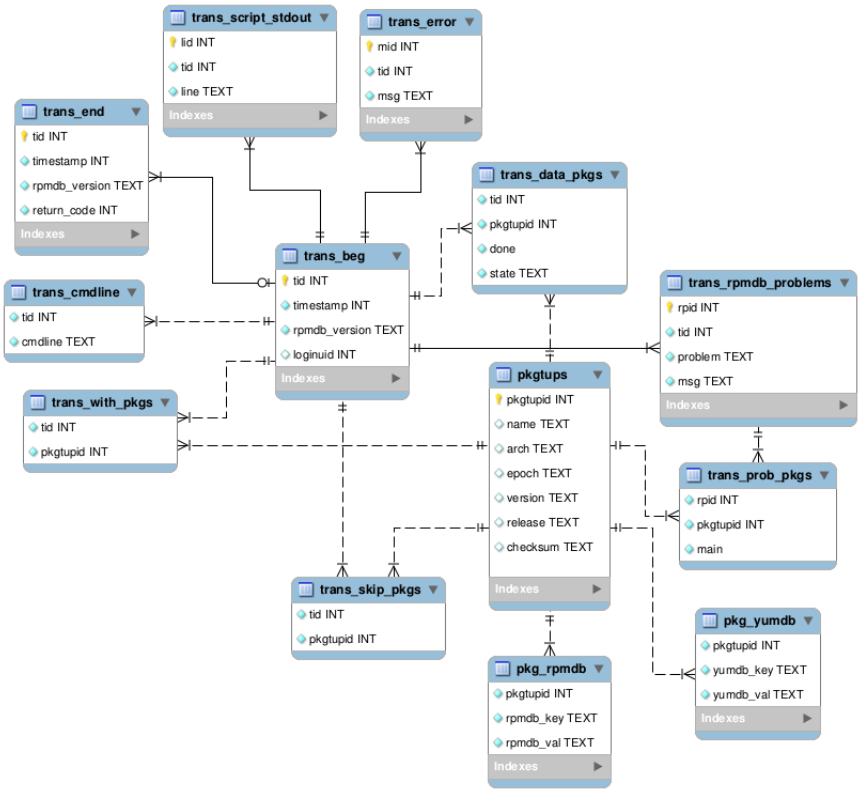
[7] Ukončený proces zjevně aktualizací ovlivněn nebude

6.2 Práce s balíčky

Především nás budou zajímat tři operace. Zjištění seznamu balíčků, které byly od daného času modifikovány, zjištění základních informací o konkrétním balíčku, jako například název, popis, atd. a nakonec zjištění seznamu souborů, které balíček poskytuje. Ukážeme si, že každou z nich lze implementovat několika různými způsoby. Objasníme, který z nich vyhovuje potřebám této práce nejvíce a proč tomu tak je.

6.2.1 Historie balíčků

Balíčkovací systém distribuce Fedora, DNF využívá pro uchovávání historie operací, prováděných s balíčky, sqlite databázi v adresáři `/var/lib/dnf/history`. Z ní se můžeme dozvědět informace o všech proběhlých transakcích. Schéma databáze vypadá následovně.



Obrázek 1: Schéma databáze balíčkovacího systému DNF

My se budeme zajímat výhradně o úspěšně proběhlé transakce, proto nás tabulky, zabývající se chybovými stavy, nebudou zajímat. Nám užitečné informace se nacházejí především v tabulkách `trans_beg`, `trans_end`, `trans_data_pkgs` a `pkgtups`. Z prvních dvou lze zjistit, kdy byly proběhlé transakce spuštěny a dokončeny. Kterých balíčků se týkali a jaké operace s nimi byly prováděny,

popisuje tabulka `trans_data_pkgs`. Dané balíčky ovšem rozlišuje pouze pomocí jejich unikátního identifikátoru `pkgtupid`. Pro zjištění více informací o daném balíčku je nutné nalézt jeho záznam v poslední zmíněné tabulce `pkgtups`.

Na základě znalostí těchto znalostí můžeme bez dalších odboček přejít přímo ke zkoumání historie balíčků. V programovacím jazyce Python máme pro práci se sqlite databázemi k dispozici balíček `sqlite3`. Ten umožní dotazování se vůči databázi pomocí vlastních SQL dotazů. Nejdříve se tedy zaměříme na vytvoření dotazu pro získání všech balíčků modifikovaných od určitého bodu v čase.

Informace o balíčcích, jako například jeho název, nebo verze, jsou uloženy v tabulce `pkgtups`. Není zde však datum jeho změny, proto musíme začít z druhého konce. Provedené transakce včetně času jejich spuštění se nacházejí v tabulce `trans_beg`. Odtud dovedeme transakce spojit s relacemi v tabulce `trans_data_pkgs` přes atribut `tid`. Výsledkem spojení budou relace obsahující atribut `pkgtupid`, pomocí jenž můžeme provést spojení s tabulkou `pkgtups`, o kterou nám z počátku šlo.
Současným mezivýsledkem jsou všechny relace vzniklé spojením zmíněných tabulek. Pomocí restrikce tuto relaci omezíme pouze na relace pro které platí, že atribut `trans_beg.timestamp` nabývá hodnot větších než t, kde t je argument ve formátu definovaném jako Unix time[8].

V jazyce SQL lze tento problém vyjádřit následovně.

```
SELECT DISTINCT pkgtups.name
FROM trans_beg JOIN trans_data_pkgs JOIN pkgtups
ON trans_data_pkgs.pkgtupid=pkgtups.pkgtupid
AND trans_data_pkgs.tid=trans_beg.tid
WHERE  trans_beg.timestamp > t
```

Zdrojový kód 7: Balíčky změněné od času t - SQL dotaz

Pokračujme spuštěním tohoto dotazu v jazyce Python a získáním jeho výsledku. Předpokládejme předchozí dotaz[9] uložený v proměnné `sql`, umístění databázového souboru v proměnné `db_file` a libovolný čas uvedeného formátu v proměnné `unix_time`. Tento zdrojový kód ukazuje, jak se lze dotázat vůči dané sqlite databázi a získat výsledek.

```
import sqlite3
conn = sqlite3.connect(db_file)
conn.row_factory = sqlite3.Row
cursor = conn.cursor()
cursor.execute(sql, [unix_time])
return cursor.fetchall()
```

Zdrojový kód 8: Balíčky změněné od času t - Python

[8]Známém také jako POSIX time, nebo Epoch time. Tento formát popisuje počet sekund uplynulých od 00:00:00 světového času dne 1. 1. 1970

[9]Nutno nahradit proměnnou t symbolem otazníku

Ukázkový kód je natolik sebevysvětlující, že se obejde bez většího komentáře. Za pozornost stojí především metoda `fetchall()`, která vrací seznam všech záznamů výsledku. Ve výchozím nastavení jsou jednotlivé záznamy reprezentovány datovou strukturou `tuple`. Jedná se o strukturu, která by se dala nejlépe popsat jako uspořádaná n-tice. Při přístupu k hodnotám atributů potom záleží na jejich pořadí v n-tici. Tento přístup má velké nevýhody, proto na třetím řádku ukázkového kódu specifikujeme, že chceme jednotlivé záznamy vracet reprezentované pomocí objektů třídy `sqlite3.Row`. To umožní přístup k atributům pomocí jejich názvů.

6.2.2 Informace o balíčku

V předchozí kapitole jsme pracovali s databázovým souborem, uchovávajícím informace o proběhlých operacích s balíčky. Odtud jsme dokázali zjistit například jejich název, verzi a pár dalších. Nicméně o balíčcích lze zjišťovat informací mnohem více. Jsou však uloženy na jiném místě[10] a proto musíme k problému přistupovat novým způsobem.
Pro zjištění všech, o balíčku dostupných, informací, můžeme použít příkaz:

```
rpm -qi nazev_balicku
```

Výpis těch zajímavějších můžeme vidět zde:

```
$[jkadlcik ~]-> rpm -qi vim-X11
Name        : vim-X11
Epoch       : 2
Version     : 7.4.179
Release     : 1.fc20
Architecture: i686
Install Date: Ne 17. srpen 2014, 17:16:24 CEST
Group       : Applications/Editors
Size        : 2472759
License     : Vim
Packager    : Fedora Project
URL         : http://www.vim.org/
Summary     : The VIM version of the vi editor for the X Window System
Description :
VIM (VIsual editor iMproved) is an updated and improved version of the
vi editor.  Vi was the first real screen-based editor for UNIX, and is
still very popular.  VIM improves on vi by adding new features:
...
```

Zdrojový kód 9: Informace o balíčku

Tyto informace lze v prostředí jazyka Python získat a zpracovat v zásadě dvěma způsoby. Prvním je použití modulu `subprocess` ke spuštění výše zmíněného příkazu a získání jeho výstupu.

[10] V databázi RPMDB umístěné v adresáři `/var/lib/rpm/`

```
>>> import subprocess
>>> cmd = ['rpm', '-qi', 'vim-X11']
>>> process = subprocess.Popen(cmd, stdout=subprocess.PIPE)
>>> out = process.communicate()[0]
>>> out = out.split('\n')
```

Zdrojový kód 10: Parsování informací o balíčku

Tato varianta však není ideální. Vyžaduje ruční parsování výstupu, jakožto textového řetězce a navíc dochází k drobnému zpomalení z důvodu spouštění nového procesu. Tato prodleva by byla zanedbatelná v případě jednorázového volání, informace však mohou být zjišťovány o velkém množství balíčků, což by vedlo k výraznému zpomalení programu.

Mnohem lepší variantou je použití aplikačního rozhraní, poskytovaného modulem `rpm`, které nabídne čistější řešení, netrpící zmíněnými neduhy.

```
>>> import rpm
>>> ts = rpm.TransactionSet()
>>> mi = ts.dbMatch(rpm.RPMTAG_NAME, "vim-X11")
>>> package_hdr = mi.next()

>>> package_hdr[rpm.RPMTAG_SUMMARY]
# The VIM version of the vi editor for the X Window System

>>> package.category = package_hdr[rpm.RPMTAG_GROUP]
# Applications/Editors
```

Zdrojový kód 11: Získání informací o balíčku pomocí rpm API

V tomto kódu se vyskytuje spousta neobjasněných elementů. Pojďme si je postupně projít a vysvětlit jejich význam. Každý skript pracující s balíčkem RPM musí zákonitě začínat instancováním třídy `TransactionSet` [10]. Ta se postará o otevření databáze balíčků a poskytne rozhraní pro dotazování se vůči ní. Tím je metoda `dbMatch` použitá na následujícím řádku. Jejím výsledkem je vždy iterátor obsahující nalezené balíčky. To, jaké balíčky budou hledány, lze ovlivnit nastavením patřičných argumentů. V případě jejich vynechání, bude vrácen iterátor množiny všech nainstalovaných balíčků. Samozřejmě lze i vyhledávat pouze balíčky splňující určitá kritéria. V ukázkovém příkladu jsou vyhledány všechny balíčky, které se jmenují `vim-X11`. Jméno balíčku je však unikátní vlastnost, takže iterátor obsahuje pouze jednu položku (případně žádnou pokud nebyla nalezena shoda). Získáme ji pomocí metody `next()`.
Jednotlivé balíčky jsou reprezentovány pomocí slovníků[11]. To znamená, že jejich klíči jsou textové řetězce `"name"`, `"summary"`, `"group"` a tak podobně. Konstanty `RPMTAG_NAME`, `RPMTAG_SUMMARY`, `RPMTAG_GROUP`, apod. slouží pouze jako tenká abstrakční vrstva, vracející daný klíč.

[11] Jiný název pro konstrukci známější především jako asociativní pole

6.2.3 Seznam souborů poskytovaných balíčkem

Každý balíček v sobě nese množinu souborů, které při instalaci nakopíruje do systému[12]. Říkejme, že balíček tyto soubory poskytuje. V případě, že dva balíčky poskytují tentýž soubor, jsou navzájem vyloučeny. To znamená, že není možné, nainstalovat je současně. O každém souboru v operačním systému lze tedy říci, že v daný moment, je poskytován nanejvýš jedním balíčkem[13].

K zjištění seznamu všech souborů poskytovaných daným balíčkem, slouží příkaz:

```
rpm -ql nazev_balicku
```

Pro získání a zpracování tohoto seznamu platí totéž, co v předchozím případě. Varianta využívající modul `subprocess`, by vypadala velmi podobně. Oproti předchozímu použití, by se změnil pouze spouštěný příkaz. Navíc jsme tuto variantu implementace z dobrých důvodů vyloučili, proto se budeme zabývat především použitím aplikačního rozhraní modulu `rpm`.

```
>>> import rpm
>>> ts = rpm.TransactionSet()
>>> mi = ts.dbMatch(rpm.RPMTAG_NAME, "vim-X11")
>>> fi = rpm.fi(mi.next())
>>> return [f[0] for f in fi]
```

Zdrojový kód 12: Získání seznamu souborů poskytovaných balíčkem

V tomto ukázkovém příkladu nově narážíme na funkci `fi` její výsledky. Tato funkce pro daný balíček vrací objekt reprezentující seznam, jím poskytovaných souborů včetně jejich atributů. Položkami tohoto seznamu jsou uspořádané n-tice, jejichž první složku tvoří název souboru.

6.3 Vyhledání ovlivněných aplikací

V předchozích kapitolách jsme si vysvětlili, jak v programovacím jazyce Python získat všechny data, které algoritmus popsaný v kapitole 3.2 na svém vstupu vyžaduje. Nyní bychom se mohli podívat na konkrétní implementaci tohoto algoritmu. Nicméně, neučiníme tak, protože samotná implementace není ničím zajímavá. Jedná se pouze o přepis pseudokódu z téže kapitoly, do zdrojového kódu jazyka Python. Z tohoto důvodu bych případné zájemce rád odkázal přímo na zdrojový kód programu.

[12] Ne tak docela každý. Existují totiž tzv. metabalíčky, které nenesou žádné soubory, ale pouze zprostředkují nainstalování jiných balíčků.

[13] Nikoli však právě jedním, protože, ne každý soubor v systému musí být vlastněn nějakým balíčkem. Protipříkladem budiž uživatelská data.

6.4 Rozšíření pro balíčkovací systém DNF

DNF poskytuje aplikační rozhraní pro tvorbu dvou druhů rozšíření. Prvním z nich je takzvaný plugin, jenž může implementovat metody, které budou provedeny v daných momentech instalačního procesu. Spustit vlastní kód tedy lze například během čtení konfiguračních souborů, před provedením plánované transakce, či ihned po ní. Vykonat lze libovolný kód, nicméně zajímavá je především možnost zkoumat prováděnou transakci a operace s balíčky, jež se chystá provést. Nemusí se však jednat pouze o čtení. Transakci lze podle vlastních potřeb libovolně upravovat. Druhý typ rozšíření umožňuje tvorbu vlastních příkazů, které lze spustit v příkazové řádce skrze program `dnf`.

Implementace vlastního příkazu potom může vypadat následovně.

```
import dnf
class FooCommand(dnf.cli.Command):
    aliases = ["foo"]
    def run(self, args):
        print "Called `dnf foo {}`".format(" ".join(args))
```

Zdrojový kód 13: Ukázka vlastního příkazu pro program `dnf`

Při spuštění, ukázkový příkaz pouze vypíše, jakým způsobem byl zavolán. Pro zprovoznění je nutné soubor uložit do adresáře definovaného v konfiguračním souboru jako `pluginpath`. Ve výchozím nastavení se jedná o adresář `/usr/lib/python2.7/site-packages/dnf-plugins/`. A dále zaregistrovat příkaz do uživatelského rozhraní, což můžeme zajistit následujícím pluginem.

```
class Foo(dnf.Plugin):
    name = "foo"
    def __init__(self, base, cli):
        super(Foo, self).__init__(base, cli)
        cli.register_command(FooCommand)
```

Zdrojový kód 14: Plugin registrující vlastní příkaz

Nyní, podíváme-li se pomocí `dnf help` do nápovědy, vidíme příkaz `foo`, který jsme vytvořili naším rozšířením. Zavoláme jej jednoduše pomocí `dnf foo arg1 arg2 arg3`.

Na závěr této vsuvkové kapitoly si ještě ukážeme plugin, který za každou úspěšně provedenou transakcí vypíše seznam balíčků, jež nainstalovala.

```
class Bar(dnf.Plugin):
    name = "foo"
    def __init__(self, base, cli):
        super(Bar, self).__init__(base, cli)
        self.base = base

    def transaction(self):
        print [p.name for p in self.base.transaction.install_set]
```

Zdrojový kód 15: Plugin vypisující seznam transakcí nainstalovaných balíčků

7 Tracer

V předchozích kapitolách jsme představili problematiku hledání aktualizacemi ovlivněných aplikací, popsali algoritmy pro jejich hledání a způsob, jak je lze implementovat v jazyce python pro linuxovou distribuci Fedora. Nyní se podíváme na projekt nesoucí název Tracer, jenž na těchto základech vznikl.

7.1 Uživatelská příručka

Následuje pouze nástin možných případů užití. Kompletní manuálovou stránku programu naleznete v příloze A.

7.1.1 Standardní použití

Primárním a zároveň nejjednodušším způsobem, jak program spustit, je příkazem[14] `sudo tracer`. Na výstupu obdržíme seznam ovlivněných aplikací, tak jak byl požadován v zadání. Ukázkový výstup můžeme vidět na následujícím příkladu.

```
$[jkadlcik  ~]-> sudo tracer
You should restart:
  * Some applications using:
      sudo service apache2 restart
      sudo service mpd restart

  * These applications manually:
      chromium
      dolphin
      gvim

Additionally to those process above, there are:
  - 6 processes requiring restarting your session
  - 2 processes requiring reboot
```

Zdrojový kód 16: Ukázkový výstup příkazu `sudo tracer`

Formát výstupu je koncipován především z pohledu uživatelské přívětivosti, proto jsou nalezené aplikace jsou rozděleny dle kategorií popsaných v kapitole 3.4. První skupinu aplikací lze restartovat pouhým zkopírováním bloku navrhovaných příkazů do terminálu.

Ve výchozím stavu se vypíše pouze poznámka o nerestartovatelných aplikacích, namísto celého jejich seznamu, protože s nimi uživatel stejně nemůže nic udělat. Z obnobného důvodu jsou vypsány pouze aplikace současného uživatele. Pro vypsání všech aplikací všech uživatelů, lze použít přepínače `--all` a `--everyone`.

[14] Oprávnění superuživatele je nutné, protože program potřebuje číst databázi balíčkovacího systému.

7.1.2 Zobrazení konkrétní aplikace

Uživatel může ve výstupu narazit na aplikaci, kterou například nezná, nebo netuší, proč je potřeba ji restartovat a jak toho docílit. Z tohoto důvodu existuje přepínač `-s` neboli `--show`, jenž zobrazí všechny informace, které Tracer o aplikaci zná a se kterými pracuje.

```
$[jkadlcik  ~]-> tracer -s apache2
* apache2
    Package:      www-servers/apache
    Description:  The Apache Web Server.
    Type:         Daemon
    State:        apache2 has been started by root 34 minutes ago. PID - 18816

    How to restart:
         service apache2 restart
```

Zdrojový kód 17: Ukázkový výstup příkazu `sudo tracer -s apache2`

Stejně jako u většiny UNIXových programů, lze množství vypisovaných informací ovlivnit přepínači `-v` a `-vv`. První varianta zobrazí i seznam balíčků, jenž aplikaci ovlivnil. Druhá varianta pak zobrazí i seznam konkrétních souborů, které balíčky aktualizovali a aplikace je využívá. Konečně, výpis těchto pomocných informací ke každé nalezené aplikaci, lze spustit přepínačem `--helpers`.

7.1.3 Interaktivní mód

V určitých případech může uživatel chtít zobrazit pomocné informace pouze k několika aplikacím, ale nechce vypisovat jejich názvy. Pro tento účel byl zaveden interaktivní režim. Pokud spustíme Tracer s parametrem `-i`, nebo `--interactive`, obdržíme na výstupu očíslovaný seznam aplikací zakončený promptem. Postupně zadáváme čísla aplikací, které nás zajímají a dostáváme výpis pomocných informací.

7.1.4 DNF plugin

Specialitou programu pro uživatele linuxové distribuce Fedora je rozšíření pro balíčkovací systém DNF, spouštějící program Tracer po každé úspěšně provedené transakci. Tedy je spuštěno automaticky po nainstalování, aktualizaci, či odstranění libovolného balíčku. Toto rozšíření nevyžaduje žádnou interakci s uživatelem a není potřeba jej nijak manuálně nastavovat. Výstup vypadá totožně s ukázkou v kapitole 7.1.1.

7.2 Licence

Program je vyvíjen pod svobodnou licencí GNU/GPLv2, jejíž plné znění naleznete v příloze B. Tato licence zajišťuje, že každý člověk má právo studovat zdrojový kód daného programu. To znamená číst jej, modifikovat a libovolně s ním experimentovat. Původní, nebo pozměněné dílo může dále distribuovat, avšak pod

podmínkou, že tyto klony budou poskytovat stejná práva, jako původní software. Licence navíc umožňuje využívat dílo k libovolnému účelu.

Při použití této licence můžeme o Traceru hovořit jako o otevřeném, či ještě lépe, svobodném software.

7.3 Verzování

Přirozeným požadavkem na software, je „rozumný“ způsob jeho verzování. Na první pohled to sice nemusí být zřejmé, ale případní uživatelé musejí být schopni určit, jakou verzi používají. A to hned z několika důvodů. Software se v průběhu času vyvíjí a uživatelé by jej měli průběžně aktualizovat[15]. K tomu je ovšem potřeba identifikovat používanou verzi a dokázat určit, zda je jiná verze novější, nebo ne. Programátoři navíc nejsou neomylní a jejich dílo může obsahovat chyby. V případě jejich hlášení je nutné uvést verzi programu, která danou chybou trpí, jinak bude velmi problematické, až nemožné, ji nalézt a opravit.

Z tohoto důvodu Tracer k verzování využívá nástroje GIT, kde je každá logická[16] úprava zdrojového kódu uložena jako popis transformačních kroků, potřebných pro převedení stávajícího kódu na nový. Krokům k provedení jedné logické změny se v kontextu nástroje GIT říká commit. Každý takový commit je označen pomocí kontrolního součtu a dle historie commitů lze určit jejich pořadí [11]. Určité milníky potom lze označit a přiřadit jim vlastní název (tag). Tím například může být číslo verze.

Člověk využívající vývojovou větev Traceru může tedy identifikovat přesnou verzi pomocí kontrolního součtu nejnovějšího commitu, případně pomocí jména tagu.

7.4 Zdrojový kód

Jakožto svobodný software, má Tracer zdrojové kódy a vše co k nim patří, veřejně vystavené. Kdokoli tak může uplatnit práva zaručená licencí GNU/GPLv2. Pro vytvoření vlastní kopie zdrojového kódu ve svém počítači, spusťte příkaz:

```
git clone https://github.com/FrostyX/tracer.git
```

Takto bude získána nejaktuálnější verze zdrojového kódu, která neprošla řádným testovacím procesem a v žádném případě ji nelze považovat za stabilní. Její užití je doporučeno pouze vývojářům programu Tracer a lidem majícím zájem o pohled na nejnovější funkcionalitu tohoto programu. Avšak použití pouze na vlastní nebezpečí.

[15] To je ostatně motivací této práce

[16] Určitá posloupnost změn tvořící pojmenovatelný celek, týkající se právě jednoho tématu

7.5 Instalační balíčky pro Fedoru

Cílovou platformou tohoto programu je linuxová distribuce Fedora. Zde je jeho zprovoznění velmi přímočaré. Nainstalovat lze buď pouze samotný program Tracer, jež bude potřeba manuálně spouštět, nebo i plugin pro balíčkovací systém DNF, který jej bude spouštět automaticky. Jsou distribuovány odděleně. Samotnou instalaci lze provést pomocí balíčkovacích systémů YUM, DNF, či jejich grafických nástaveb. Následující ukázka vysvětlí instalaci skrze balíčkovací systém DNF.

```
# Instalace pouze programu Tracer
sudo dnf install tracer

# Instalace včetně DNF pluginu
# Netřeba explicitně instalovat balík 'tracer'
sudo dnf install dnf-plugins-extras-tracer
```

Zdrojový kód 18: Instalace programu Tracer pomocí balíčkovacího systému DNF

Tyto příkazy zajistí stažení a instalaci balíčků, včetně všech potřebných závislostí. Získány budou z oficiálního repozitáře distribuce Fedora. Pravděpodobně tedy nebude nainstalována stejná verze programu, jako ta, která se nachází na přílohovém CD k této práci.

Přibalenou verzi lze nainstalovat spuštěním následujících příkazu:

```
# Přesun do adresáře 'install' na přiloženém CD
cd /run/media/<user>/<drive>/install/

# Instalace pouze programu Tracer
sudo dnf install ./tracer-0.6.2-1.fc22.noarch

# Instalace včetně DNF pluginu
# Netřeba explicitně instalovat balík 'tracer'
sudo dnf install ./python-dnf-plugins-extras-tracer-0.0.9-1.fc22.noarch
```

Zdrojový kód 19: Instalace přiložené verze programu Tracer

7.6 Dokumentace

Původní dokumentace projektu byla kolekce souborů oddělená od zdrojového kódu programu a uživatelům byla poskytována prostřednictvím wiki systému. Nevýhodu tohoto řešení jsem si uvědomil záhy po jeho nasazení. Ve chvíli, kdy je dokumentace oddělena od zdrojového kódu, nastává problém s rozlišováním, pro kterou verzi kódu je určena. Jde-li o vydanou stabilní verzi, lze ji samozřejmě specifikovat někde v textu. Nicméně v případě vývojové větvě lze těžko určit, od kterého commitu popisovaná funkcionalita funguje.

Mnohem lepším řešením je vedení dokumentace přímo v adresáři zdrojového kódu programu. O oboje se pak stará stejný verzovací systém a tím pádem je k dispozici dokumentace věrně odpovídající zdrojovému kódu v libovolném okamžiku.

Projekt nyní využívá nástroj Sphinx, který na základě souborů ve formátu rst[17] dokáže vygenerovat dokumentaci v několika výstupních formátech, jako například HTML, či PDF. Do jednotlivých stránek navíc umožňuje vkládat konkrétní entity zdrojového kódu. Odkaz na webovou dokumentaci projektu naleznete v kapitole 7.9. Vlastní verzi si můžete vygenerovat pomocí příkazů[18]:

```
cd doc
make help          # Pro výběr cílového formátu
make <target>
```

Zdrojový kód 20: Vygenerování dokumentace pomocí nástroje sphinx

7.7 Lokalizace

Přestože projekt obsahuje velmi jednoduché uživatelské rozhraní a nejspíš jej lze používat i s elementárními znalostmi angličtiny, poskytuje možnost jazykových lokalizací. Součástí zdrojových kódů je soubor `tracer.pot` obsahující všechny lokalizovatelné řetězce programu a dalé pak soubory s příponou `.po`, obsahující překlady těchto řetězců do konkrétních jazyků. Tyto soubory jsou vyvěšeny na překladatelském serveru http://transifex.com.

Při vytváření instalačního balíčku se ze všech `.po` souborů vygenerují jejich binární verze s příponou `.mo`. Ty jsou při instalaci balíčku kopírovány do systémového adresáře s překlady. V závislosti na uživatelem preferovaném jazyku systému, je při spuštění vybrána kýžená lokalizace. V současné době existuje kompletní překlad pouze do českého jazyka, přičemž ostatním uživatelům bude zobrazena výchozí anglická verze.

7.8 Programátorská příručka

Tato kapitola je určená především novým vývojářům a všem zájemcům, kteří chtějí nějakým způsobem pracovat se zdrojovými kódy projektu. Objasní jejich organizaci, použitá paradigmata a poradí, jak ověřit, že jimi provedené změny neměly nezamýšlený, nežádoucí efekt.

7.8.1 Architektura zdrojového kódu

V domovském adresáři projektu najdeme pouze soubory potřebné k balíčkování a konfigurační soubory pro služby, jež projekt využívá. Dále vidíme několik adresářů, z nichž zajímavé jsou především tyto následující.

Spouštěcí soubor programu nalezneme v adresáři `bin`. Tento soubor obsahuje pouze nejnutnější minimum zdrojového kódu, nutného k zavedení programu.

[17] reStructuredText – plaintextový formát umožňující intiutivní strukturování dokumentu
[18] Vyžaduje instalaci balíčku `python-sphinx`

Naopak naprostá většina kódu se nachází v adresáři `tracer`, který je při instalaci nakopírován k systémovým modulům jazyka Python.

Tento kód je povětšinou objektově orientovaný a koncepčně je rozdělen do jednotlivých celků architektury Model-View-Controller (MVC) [12]. Šablony (View), nejsou tvořeny pomocí speciálního šablonovacího systému, jako tomu často bývá například u webových aplikací, nýbrž jde o obyčejné soubory obsahující zdrojový kód jazyka Python. Slouží však pouze k vypsání výstupu na požadované zařízení. Kontrolery se starají o naplnění těchto šablon „vypočítanými“ daty. Největším celkem jsou modely tvořící zbytek adresáře. Ty slouží ke zjišťování informací o systému, procesech, balíčcích, zajišťují jazykovou lokalizaci a jiné.

7.8.2 Testování

Zdrojový kód projektu je prověřován pomocí jednotkových testů, i testů uživatelského rozhraní. Oboje jsou realizovány pomocí modulu `unittest`. Testy však nepokrývají každou jednotlivou řádku kódu a negarantují stoprocentní spolehlivost, nicméně za určitých okolností, například při refaktorování, jsou velmi nápomocné. Všechny testy jsou pomocí CI spouštěny automaticky při uložení libovolné změny do verzovacího systému. Ručně, v příkazové řádce, lze testy spustit například pomocí nástrojů `nosetests`, nebo `unit2`.

7.9 Odkazy

V tuto chvíli bych rád uvedl přehledovou tabulku s odkazy na všechny důležité služby, které projekt využívá.

Homepage:	http://tracer-package.com
Dokumentace:	http://docs.tracer-package.com
Zdrojový kód:	https://github.com/FrostyX/tracer
Travis CI:	https://travis-ci.org/FrostyX/tracer
Transifex (lokalizace):	https://www.transifex.com/organization/frostyx/dashboard/tracer

8 Srovnání s konkurencí

Rád bych se pokusil objektivně porovnat svou práci s konkurenčními projekty. Během počátečního průzkumu, než jsem se vůbec začal touto problematikou zabývat do hloubky, hledal jsem již existující projekt, který bych mohl podpořit, či na něj navázat. Několik jsem jich našel, avšak žádný z nich nebyl dostatečně vyhovující a tak vznikl Tracer. V průběhu jeho vývoje jsem postupně narazil na několik dalších, existujících nástrojů, které vznikly teprve nedávno.

Nyní si vybrané z nich projdeme, zaměříme se na jejich cíle, vlastnosti, podporované operační systémy, uživatelské rozhraní a podobně.

8.1 needs-restarting

Nástroj `needs-restarting` je od roku 2009 součástí balíčku `yum-utils`. Viz commit `43e630f` v repozitáři git://yum.baseurl.org/yum-utils.git. Jak napovídá samotný název balíčku, jedná se o plugin pro zastaralý balíčkovací systém YUM.

```
$[jkadlcik ~]$ sudo needs-restarting
623 : /usr/libexec/accounts-daemon
1581 : kdeinit4: kded4 [kdei e
1327 : /usr/bin/kglobalaccel5
597 : /sbin/auditd -n
741 : /usr/sbin/VBoxService
705 : /usr/lib/polkit-1/polkitd --no-debug
1339 : /usr/libexec/kf5/kscreen_backend_launcher
1140 : (sd-pam)
1229 : /usr/bin/VBoxClient --clipboard
1230 : /usr/bin/VBoxClient --clipboard
1488 : /usr/bin/pulseaudio --start --log-target=syslog
```

Zdrojový kód 21: Ukázka výstupu nástroje `needs-restarting`

8.2 checkrestart

Obdobným nástrojem určeným pro linuxovou distribuci Debian je nástroj `checkrestart` z balíčku `debian-goodies`.

```
$[jkadlcik needrestart]-> checkrestart
Found 5 processes using old versions of upgraded files
(4 distinct programs)
(4 distinct packages)
These processes do not seem to have an associated init script to restart them:
php5-cli:
        11435   /usr/bin/php5
libkactivities-bin:
        21363   /usr/bin/kactivitymanagerd
mysql-server-core-5.5:
        10726   /usr/sbin/mysqld
        21380   /usr/sbin/mysqld
akonadi-server:
        10724   /usr/bin/akonadiserver
```

Zdrojový kód 22: Ukázka výstupu nástroje `checkrestart`

8.3 needrestart

Univerzálnějším nástrojem podporujícím balíčkovací systémy DPKG, RPM a Pacman je `needrestart`. Tento projekt vznikl v březnu roku 2013. Viz commit `6889143` v repozitáři https://github.com/liske/needrestart.git. Tento program bohužel nemá instalační balíček pro distribuci Fedora.

Původním cílem, který přetrval až do verze 2.0, tedy po většinu života tohoto projektu, bylo vyhledání pouze systémových služeb. Avšak nyní disponuje velmi zajímavými vlastnostmi jako například podporou vyhledávání v kontejnerech, kontrolou jádra systému, podporou pro interpretované aplikace, či možností navázat automatické spuštění programu po spuštění daného balíčkovacího systému.

```
$[jkadlcik needrestart]-> ./needrestart
Scanning processes...
Services to be restarted:
Restart NetworkManager.service? [yNas?] n
Restart accounts-daemon.service? [Ynas?] n
Your outdated processes:
bash[2677], chrome[18513, 2249, 15420], dbus-daemon[1877, 1956], ...
```

Zdrojový kód 23: Ukázka výstupu nástroje `needrestart`

8.4 Shrnutí

Nejdříve, `checkrestart` je určen výhradně pro distribuci Debian a jeho deriváty. Navíc, subjektivně mi přijde jeho uživatelské rozhraní jako nejhorší ze všech zkoumaných nástrojů. Jeho alternativou pro distribuci Fedora doposud byl `needs-restarting`, který je ale nyní zastaralý kvůli plánovanému odstranění YUMu z distribuce. Mým cílem je zaujmutí uživatelů tohoto programu a jejich následný přeliv k Traceru.

Oproti tomu `needrestart` je velmi perspektivní projekt, který stále rozšiřuje svou funkcionalitu. Tracer ji kdysi dokázal použitím přepínače `--services-only` pokrýt téměř celou, nicméně aktuální verze `needrestart` poskytuje například podporu kontejnerů, nebo kontrolu jádra systému, kterou Tracer zatím neumí. Tento nástroj mě oslovil a nejspíš bych se připojil k jeho vývoji namísto založení nového projektu. Bohužel `needrestart` i Tracer vznikly oba v roce 2013 a po většinu vývoje koexistovali ve vzájemné anonymitě. Největší nevýhodou tohoto nástroje vidím v chybějícím balíčku pro distribuci Fedora.

Určit nejlepší nástroj není možné a vždy bude záležet na konkrétních preferencích daného uživatele. Program `needrestart` na rozdíl od všech ostatních sází na interaktivní režim ve výchozím stavu, nicméně podporuje i režim pouhého výpisu pomocí přepínače `-r l`. Ovšem i v tomto režimu vyžaduje interakci s uživatelem pro potvrzení neaktuálního kernelu. To může být v určitých situacích poněkud nepříjemné. Tracer k uživatelskému rozhraní přistupuje obráceně a vyžaduje interakci s uživatelem pouze při použití přepínače `-i`. Ostatní konkurenční nástroje podporují pouze režim výpisu, přesto však dovedou být rozdílné. Program `needs-restarting` rozlišuje vypsané procesy pomocí jejich identifikátoru PID a může se tedy stát, že bude vypsáno několik, stejně se jmenujících, procesů. Podobně tomu je i u `checkrestart`, který tento seznam navíc obohatí o název balíčku, který ovlivnil dané procesy.

9 Život projektu Tracer

Projekt Tracer spatřil světlo světa v říjnu roku 2013. Už v této době poskytoval většinu současné funkcionality, avšak se v průběhu let výrazně vyvinul. Doposud vyšlo 22 verzí, přičemž nejnovější nese označení 0.6.1. Když vývoj započal, používala se Fedora 19 a jazyk Python byl masivně rozšířený ve verzi 2.7. Nyní jsme se posunuli k Fedoře 22, která je poslední, jež používá tuto verzi Pythonu jako výchozí. Tracer se musel přizpůsobit a nyní je připraven běžet ve Fedoře 23 na Pythonu 3.x. Přištím Fedoru také čeká další významná změna, kterou je nahrazení původního balíčkovacího systému YUM novým nástrojem DNF. Tracer původně fungoval výhradně s YUMem, avšak jak DNF začal získávat na důležitosti, projekt začal podporovat oba systémy současně, aniž by se o to musel uživatel zajímat. Nové verze však od podpory YUMu ze zjevných důvodů upustí.

Zajímavé je také ohlédnutí na způsob distribuce tohoto programu. Od svého počátku byl projekt umístěn ve veřejném repozitáři na GitHubu a uživatelé si mohli zdrojové kódy několika způsoby stáhnout. Tento způsob byl vhodný pouze zpočátku, protože uživatelé samořejmě nechtějí instalovat a spravovat závislosti manuálně. V brzké době se proto přešlo na distribuci pomocí vlastního repozitáře v systému Copr[19]. Tato varianta přetrvala dodnes, avšak slouží pouze k vytváření testovacích balíčků. Stabilní verze jsou distribuovány skrze oficiální softwarové repozitáře distribuce Fedora. Autor této práce se stal „balíčkářem“ distribuce a o balíček nástroje Tracer se stará osobně. Plugin pro systém DNF se podařilo připojit k oficiálně podporovaným komunitním pluginům a jeho distribuci zajišťuje správce této sbírky.

Od samotného počátku nebyla očekávána velká uživatelská základna. Mnoho lidí vůbec netuší, že nástroj tohoto typu vůbec potřebuje a nezajímá je to. Navíc projekt cílí především na uživatele RPM distribucí. Počet instalací, potažmo současný počet uživatelů, nelze přesně určit, protože Fedora neudržuje statistiku stažení jednotlivých balíčků. Na základě jistého problému[20] postihujícího všechny uživatele Traceru na Fedoře 21, však lze alespoň určit spodní hranici, již tvoří 240 uživatelů, kteří problém ohlásili. Takovou uživatelskou základnu považuji za úspěch. V rámci zpětné vazby od uživatelů bylo nahlášeno celkem 46 chyb a žádostí o rozšíření funkcionality, z nichž 36 bylo již vyřešeno. Dále bylo vyhověno třem žádostem o portaci na jiné distribuce.

[19] Copr (Cool Other Package Repo) slouží k vytváření repozitářů pro balíčky třetích stran
[20] https://bugzilla.redhat.com/show_bug.cgi?id=1187763

Závěr

Cílem mé práce bylo rozebrat problematiku „aktualizacemi ovlivněných aplikací“, konkrétně pak jejich nalezení a doporučení nejlepšího způsobu, jak docílit jejich restartování. V teoretické části jsem objasnil důležitost tohoto problému a důsledky z něho plynoucí. Pokud by si čtenář z tohoto textu odnesl pouze jedinou informaci, měla by to být právě tato. Přestože svůj systém aktualizuje, neznamená to, že se změny automaticky projeví v právě spuštěných aplikacích. Je potřeba je restartovat. Dále jsem zkoumanou problematiku rozdělil na dílčí celky a navrhl algoritmy pro jejich řešení.

Součástí práce bylo v jazyce Python implementovat software pro vyhledávání ovlivněných aplikací na linuxové distribuci Fedora. Zde jsem využil vyzkoumané algoritmy zmíněné výše. V této části bylo zajímavé především získání dat potřebných pro jejich vstupy. Čtenář se dozvěděl, jakým způsobem zjišťovat informace o spuštěných procesech, nainstalovaných balíčcích, či balíčkovacím systémem provedených transakcích.

Původní cílovou platformou byla distribuce Fedora 19 obsahující interpret Pythonu verze 2.7 a balíčkovací systém YUM. V průběhu práce se všechny tyto komponenty staly neaktuálními a proto se projekt přizpůsobil současnému stavu věcí. Fedora je nyní v téměř finální fázi přechodu na systém DNF, v tuto chvíli je již YUM označen za zastaralý a v příštím vydání bude zahozen úplně. Obdobně je tomu v situaci interpreterů Pythonu. Již v příštím vydání bude výchozí jeho třetí verze namísto druhé. Můj projekt tento vývoj reflektuje a je možné jej používat, jak s komponentami, pro které byl určen původně, tak s těmi aktuálními.

Poslední částí mé práce bylo objasnit, jakým způsobem lze rozšiřovat funkcionalitu balíčkovacího systému DNF. Tu považuji za nejméně důležitou, protože pouze rekapituluje již známé a popsané informace. Z tohoto důvodu jí byla věnována pouze jedna kapitola shrnující množinu elementárních znalostí nutných k porozumění zbytku práce.

Software se v průběhu času neustále vyvíjí a vždy jej lze vylepšovat a rozšiřovat o novou funkcionalitu. V případě programu Tracer by například šlo využít nástroje PackageKit, který by mohl přinést podporu pro většinu linuxových distribucí. Má snaha bude nadále směřovat také k zahrnutí programu do výchozí instalace distribuce Fedora, aby si problematiku probíranou v této práci, začalo uvědomovat více uživatelů.

Conclusions

The goal of my thesis was to analyze the topic of „applications affected by upgrade“, particularly of their determination and suggesting the best way of achieving thier restart. In the theoretical part I explained the relevance of this topic and it‘s consequences. If the reader is supposed to learn at least one information from this thesis, it should be this one. Even though he updates his system the changes will not take effect in running applications. It is needed to restart them. Hereafter I divided given topic to sub-units and suggested algorithms for solving them.

Part of my work was to implement the software for determining affected applications on linux distribution Fedora, in Python language. Here I utilized algorithms mentioned above. In this part it was mainly interesting to get the data needed on their inputs. The reader learned how to gain information about running processes, installed packages or transactions made by package manager.

The initial targeted platform was distribution Fedora 19 with interpreter Python in version 2.7 and package manager YUM installed. During the thesis all of these components became outdated and therefore the project adapted for recent state of things. Fedora is now at final phase of conversion to the DNF manager. At this moment, YUM has been marked as outdated and in the following release it will be removed completely. The Python interpreter is in similar situation. In the following release it‘s third version will be set as default instead of the second version. My project reflects this development and it is possible to use it with the components which it was originally designed for and even with the current components.

The last part of my work was to clarify how it is possible to extend the functionality of the DNF package manager. That I consider to be the least important thing because it just sums up already known and described information. Therefore only one chapter was dedicated to wrap up the set of elementary knowledge needed for understanding the rest of the thesis.

Software changes constantly and it is always possible to improve it and extend it with new functionality. In case of Tracer program it should be possible to use the PackageKit tool which could bring the support for the most of linux distributions. I will make my effort to have the program included in the default installation of Fedora distribution in order to make more people aware of the topic described in this thesis.

A Manuálová stránka programu Tracer

Tracer determines which applications use outdated files and prints them. For special kind of applications such as services or daemons, it suggests a standard command to restart it. Detecting whether file is outdated or not is based on a simple idea. If application has loaded in memory any version of a file which is provided by any package updated since system was booted up, tracer consider this application as outdated.

A.1 Options

A.1.1 General

```
--version               print program version
-h, --help              show this help message and exit
-q, --quiet             do not print additional information
-v, --verbose           print more informations. Use -v or -vv
```

A.1.2 Modes

```
--helpers               not list applications, but list their helpers

-i, --interactive       run tracer in interactive mode. Print numbered
                        applications and give helpers based on numbers

-s app_name [app_name ...], --show app_name [app_name ...]
                        show helper for given application

-a, --all               list even session and unrestartable applications

--daemons-only, --services-only
                        list only daemons/services

--hooks-only            do not print traced applications, only run their hooks

-t TIMESTAMP, --timestamp TIMESTAMP
                        since when the updates should be

-n, --now               when there are specified packages, dont look for time
                        of their update. Use "now" instead
```

A.1.3 Users

```
-u username, --user username
-r, --root
-e, --everyone
```

A.1.4 Debug

```
--show-resource=<option>
                    options: packages | processes | rules |
                             | applications | system
                    dump informations that tracer can use
```

A.2 Examples

```
Show your applications which needs restarting (basic usage)
    tracer

Show informations about application
    tracer --show mysqld

Show even affected files of the application
    tracer --show mysqld -vv

In interactive mode show all applications modified only
through packages changed since timestamp
    tracer -iat 1414248388.04
```

B Licence GNU/GPLv2

Copyright © 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in

the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly

through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

End of Terms and Conditions

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> one line to give the program's name and a brief idea of what it does.
> Copyright (C) yyyy name of author
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) yyyy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

C Obsah přiloženého CD/DVD

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/tracer
: Kompletní zdrojové kódy programu Tracer včetně spustitelného souboru `bin/tracer.py` a programové dokumentace v podadresáři `doc/`

src/dnf-plugins-extras
: Zdrojové kódy sbírky rozšíření `dnf-plugins-extras` obsahující mimo jiné soubory `plugins/tracer.py` a `doc/tracer.rst`

install
: Adresář obsahující instalační balíčky programu Tracer a jeho DNF pluginu pro Fedoru 22

Literatura

[1] ŠILHAN, Jan. *Yum is dead, long live DNF* [online]. [cit. 2015-08-04]. Dostupný z: ⟨http://dnf.baseurl.org/2015/05/11/yum-is-dead-long-live-dnf/⟩.

[2] DNF TEAM. *DNF API Reference* [online]. [cit. 2015-08-04]. Dostupný z: ⟨https://dnf.readthedocs.org/en/latest/api.html⟩.

[3] SCHOFIELD, Ed. *Cheat Sheet: Writing Python 2-3 compatible code* [online]. [cit. 2015-08-04]. Dostupný z: ⟨http://python-future.org/compatible_idioms.html⟩.

[4] PYTHON SOFTWARE FOUNDATION. *Python 2.7.10 documentation* [online]. [cit. 2015-08-04]. Dostupný z: ⟨https://docs.python.org/2.7/⟩.

[5] PYTHON SOFTWARE FOUNDATION. *Python 3.3.6 documentation* [online]. [cit. 2015-08-04]. Dostupný z: ⟨https://docs.python.org/3.3/⟩.

[6] PETERS, Tim. *PEP 20 – The Zen of Python* [online]. [cit. 2015-08-04]. Dostupný z: ⟨https://www.python.org/dev/peps/pep-0020/⟩.

[7] ROSSUM van, Guido. *PEP 3103 – A Switch/Case Statement* [online]. [cit. 2015-08-04]. Dostupný z: ⟨https://www.python.org/dev/peps/pep-3103/⟩.

[8] ROSSUM van, Guido; WARSAW, Barry; COGHLAN, Nick. *PEP 0008 – Style Guide for Python Code* [online]. [cit. 2015-08-04]. Dostupný z: ⟨https://www.python.org/dev/peps/pep-0008/⟩.

[9] RODOLA', Giampaolo. *psutil documentation* [online]. [cit. 2015-08-04]. Dostupný z: ⟨https://pythonhosted.org/psutil/⟩.

[10] *Querying the RPM database*. Dostupný z: ⟨http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch16s03s02.html⟩.

[11] CHACON, Scott. *Pro Git*. 2010. 267 s.

[12] MASOVER, Steve. *Model-View-Controller: A Design Pattern for Software* [online]. [cit. 2015-08-04]. Dostupný z: ⟨https://ist.berkeley.edu/as-ag/pub/pdf/mvc-seminar.pdf⟩.

[13] ŠVEC, Jan. *Učebnice jazyka Python: (aneb Létající cirkus)*. 2.2, 2007. 96 s.

[14] DNF TEAM. *DNF, the next-generation replacement for Yum* [online]. [cit. 2015-08-04]. Dostupný z: ⟨https://dnf.readthedocs.org⟩.