VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

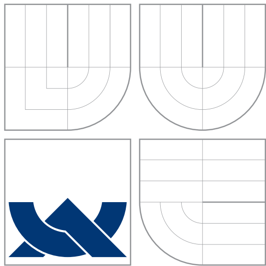
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ÚLOŽIŠTĚ GENTOO PORTAGE JAKO SOUBOROVÝ SYSTÉM ZALOŽENÝ NA RELAČNÍ DATABÁZI

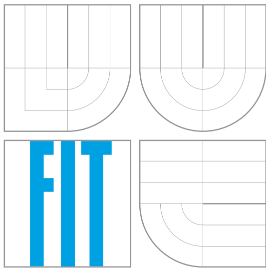
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ADAM ŠTULPA
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ÚLOŽIŠTĚ GENTOO PORTAGE JAKO SOUBOROVÝ SYSTÉM ZALOŽENÝ NA RELAČNÍ DATABÁZI

A FILE SYSTEM IMPLEMENTING STORAGE FOR GENTOO PORTAGE BASED ON A RELATIONAL DATABASE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ADAM ŠTULPA
AUTHOR

VEDOUCÍ PRÁCE — Mgr. MAREK RYCHLÝ
SUPERVISOR

BRNO 2008

// Abstrakt

Práce ze zabývá implementací programu, který pomocí knihovny FUSE dokáže zpřístupnit data v relační databázi jako klasické úložiště Gentoo Portage. Čtenář je nejdříve seznámen s jeho strukturou. V práci je popsána analýza požadavků na program a dabázi, dále jejich návrh a v předposlední kapitole je popsána implementace programu. Závěr práce hodnotí dosažené výsledky včetně srovnání s klasickým souborovým systémem a popisuje další možnosti vývoje projektu.

Klíčová slova

Linux, Gentoo Linux, Portage, FUSE, Portage, C++, Virtualní souborový systém, Systém balíčků, Relační databáze

Abstract

The thesis deals with an implementation of a program, which can make the data in relational database available as the standard storage Gentoo Portage with the assistance of FUSE library. The reader is initially familiarised with its structure. There is an analysis of the requirements on the program and the database described in the work. Furthermore, their proposal is given and in the penultimate chapter, there is described the implementation of the program. In the conclusion, there are all the outcomes evaluated including the comparison with the standard file system and further development possibilities of the project are also described.

Keywords

Linux, Gentoo Linux, Portage, FUSE, Portage, C++, Virtual File System, Package Manager, Relational Database

Citace

Adam Štulpa: Úložiště Gentoo Portage jako souborový systém založený na relační databázi, bakalářská práce, Brno, FIT VUT v Brně, 2008

Úložiště Gentoo Portage jako souborový systém založený na relační databázi

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Mgr. Marka Rychlého

. . . .
Adam Štulpa
9. května 2008

Poděkování

Rád bych poděkoval Mgr. Markovi Rychlému za rady a odborné vedení při psaní této bakalářské práce a dále své rodině za podporu při mém studiu.

© Adam Štulpa, 2008.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod 3

2 Teorie 4
- 2.1 Linux . . . 4
  - 2.1.1 Historie . . . 4
  - 2.1.2 Licence . . . 5
- 2.2 Gentoo Linux . . . 5
  - 2.2.1 Historie . . . 5
- 2.3 Portage . . . 6
  - 2.3.1 Struktura portage stromu, organizace dat . . . 6
  - 2.3.2 Ebuild . . . 6
  - 2.3.3 USE proměnné . . . 7
  - 2.3.4 Maskování a klíčová slova . . . 7
  - 2.3.5 Eclass . . . 7
  - 2.3.6 Overlays . . . 8
  - 2.3.7 Výhody a nevýhody . . . 8
- 2.4 Souborový systém . . . 8
  - 2.4.1 FUSE (Filesystem in Userspace) . . . 8
- 2.5 Relační databáze . . . 9
  - 2.5.1 Jazyk SQL . . . 10
  - 2.5.2 PostgreSQL . . . 10

3 Praktická část 11
- 3.1 Analýza . . . 11
  - 3.1.1 Požadavky na program . . . 11
  - 3.1.2 Ukládaná data . . . 11
  - 3.1.3 Způsob uložení ebuildů v databázi . . . 11
  - 3.1.4 Další soubory a adresáře . . . 14
  - 3.1.5 Data v adresáři distfiles . . . 14
  - 3.1.6 Velikost souborů ukládaných v databázi . . . 14
  - 3.1.7 Implementované funkce v knihovně FUSE . . . 14
  - 3.1.8 Přístupová práva k souborům . . . 15
- 3.2 Návrh . . . 15
  - 3.2.1 Výběr programovacího jazyka . . . 15
  - 3.2.2 Výběr databáze . . . 15
  - 3.2.3 Návrh databáze . . . 16
  - 3.2.4 Návrh programu . . . 18
- 3.3 Implementace . . . 23

3.3.1 Logování událostí . . . 24
3.3.2 Modul Core . . . 25
3.3.3 Třída Portage . . . 25
3.3.4 Ostatní třídy . . . 25
3.3.5 Vytváření nového adresáře v kořenovém adresáři . . . 25
3.3.6 Vytváření nového adresáře v adresáři skupiny . . . 25
3.3.7 Connection pool . . . 25
3.3.8 Odkazy na adresáře v souborovém systému . . . 26
3.4 Konfigurační soubory . . . 26
3.5 Přeložení programu . . . 27
3.6 Spuštění programu . . . 27
3.7 Specifikace testů . . . 27
3.7.1 Měření výkonu . . . 28

4 Závěr **29**

Kapitola 1

Úvod

Gentoo je relativně mladá linuxová distribuce, která si však získala mnoho příznivců a to zejména svým netradičním způsobem instalace software. Netradiční proto, že drtivá většina ostatních distribucí používá při instalaci již předkompilované balíčky narozdíl od Gentoo Linuxu, který klade důraz na kompilaci ze zdrojových kódů a maximální konfigurovatelnost. Každý uživatel si tedy může svůj systém „ušít“ přímo na míru.

Instalaci programů v Gentoo Linuxu zajišťuje balíčkovací systém, který se jmenuje Portage. Strom Portage je hierarchicky uspořádaná adresářová struktura. Jsou v ní uloženy ebuildy, což jsou soubory obsahující informace o tom, kde stáhnout zdrojové kódy, jak je přeložit a jak nainstalovat program do systému. Data v tomto stromu (ebuildy a další soubory včetně jejich obsahu) jsou organizována podle předem daných pravidel. Je tedy možné je uložit do databázových tabulek, což je cílem této bakalářské práce – použít místo klasického souborového systému relační databázi pro uložení tohoto stromu.

Bakalářská práce je rozdělena do čtyř kapitol. První kapitola seznamuje čtenáře s obsahem bakalářské práce. Druhá kapitola uvádí do problematiky a popisuje současný stav. Ve třetí kapitole je popsána analýza, návrh a implementace programu. Ve čtvrté kapitole je celkové zhodnocení výsledků a popsán možný směr dalšího vývoje.

Kapitola 2

Teorie

2.1 Linux

Linux je jádrem (kernelem) několika počítačových operačních systémů. Je známým příkladem svobodného softwaru a vývoje open source softwaru na rozdíl od proprietárních operačních systémů jako Microsoft Windows či Mac OS je celý jeho zdrojový kód volně k dispozici pro veřejnost a kdokoli jej může svobodně používat, upravovat a dále distribuovat. Ačkoliv termín Linux značí Linuxové jádro, často se používá pro označení celých unixových operačních systémů (známých jako GNU/Linux), které sestávají z Linuxového jádra a zároveň z knihoven a nástrojů z projektu GNU, ale i z dalších zdrojů. V nejširším významu GNU/Linuxová distribuce uceleně spojuje základní systém s velkým balíkem aplikačního softwaru, a navíc často zajišťuje uživatelsky přívětivou instalaci a následné aktualizace. Zpočátku byl Linux vyvíjen a používán zejména jednotlivými nadšenci. Časem ale získal podporu velkých společností jako IBM, Hewlett-Packard a Novell pro využití na serverech, a poslední dobou získává popularitu i na desktopovém trhu. Zastánci a analytici připisují jeho úspěch nezávislosti na dodavateli, nízkých nákladech, flexibilitě, bezpečnosti a spolehlivosti. Linux byl původně vyvíjen pro počítače s procesory architektury i386 (tedy 80386 a kompatibilními). Dnes ale podporuje všechny populární počítačové architektury i mnoho z těch méně obvyklých. Používá se v řadě zařízení od embedded systémů (jako mobilních telefony, roboti či multimediální přehrávače) přes osobní počítače až po superpočítače [3].

2.1.1 Historie

Linux je operační systém, který byl původně vytvořen studentem Linusem Torvaldsem na University of Helsinki ve Finsku. V roce 1983 založil Richard Stallman projekt GNU, který nyní poskytuje základní část většiny linuxových systémů (viz pojem GNU/Linux níže). Cílem projektu GNU je vyvinout kompletní unixový operační systém složený výhradně ze svobodného softwaru. Začátkem 90. let byly v rámci projektu GNU vytvořeny a shromážděny všechny potřebné součásti - knihovny (především glibc), překladač (GCC), textový editor (Emacs), shell (bash) a další software, ovšem kromě nejnižší úrovně, tedy jádra.V roce 1990 začal projekt GNU vyvíjet své vlastní jádro jménem GNU Hurd[12] (po upuštění od předchozího pokusu jménem Trix). Podle Thomase Bushnella, původního architekta Hurdu, bylo původním plánem přepracovat jádro BSD 4.4-Lite. Přesto se Stallman, kvůli špatné spolupráci programátorů z Kalifornské univerzity v Berkeley, rozhodl použít mikrojádro Mach. Ale, jak se později ukázalo, vývoj byl nečekaně složitý a pokračoval velmi pomalu. Mezitím v roce 1991 započal vývoj jiného jádra, které nakonec dostalo jméno „Li-

nux". Původně ho začal psát finský student helsinské univerzity Linus Torvalds jako svůj koníček. Torvalds vycházel z Minixu, což byl zjednodušený klon Unixu napsaný Andrewem Tanenbaumem pro účely výuky návrhu operačních systémů. Avšak Tanenbaum nikomu nedal svolení k úpravám svého systému, a tak Torvalds napsal vlastní náhradu Minixu. Linux začal jako emulátor terminálu napsaný v IA-32 assembleru a jazyce C. První verze linuxového jádra (0.01) byla vydána na Internetu 17. září 1991, další následovala v říjnu téhož roku. Od té doby se na tomto projektu podílely tisíce vývojářů z celého světa[3].

2.1.2 Licence

Linuxové jádro a mnoho dalších GNU komponent je licencováno pod GNU General Public License (GPL). Zdrojové kódy software pod GPL mohou být svobodně upravovány a používány, šířeny však musí být opět pod GPL (jestliže se je rozhodnete dále šířit) avšak binární formy software používající GPL mohou být poskytovány za libovolně vysokou úplatu. Ostatní subsystémy mohou mít jiné licence, ale všechny spadají do kategorie svobodného softwaru/open source. Některé knihovny například mají volnější licenci LGPL a X Window System používá MIT licenci. Ochrannou známku Linux (č. 1916230) na „software počítačového operačního systému, který usnadňuje práci s počítačem" vlastní Linus Torvalds. Licencování této ochranné známky nyní obstarává Linux Mark Institute (LMI)[3].

2.2 Gentoo Linux

Gentoo je svobodný operační systém založený na Linuxu nebo FreeBSD, který může být automaticky optimalizován a upraven na míru podle konkrétního účelu nebo potřeb. Od ostatních distribucí se liší zejména balíčkovacím systémem Portage. Je založena téměř výhradně na zdrojových kódech a jedná se o takzvanou source nebo meta-distribuci. Díky kompilaci ze zdrojových kódů je možné celý systém optimalizovat pro danou architekturu. Je spravována nadací Gentoo Foundation, která také vlastní všechna autorská práva. Na vývoji pracuje přes 300 vývojařů

2.2.1 Historie

Daniel Robbins se rozhodl vytvořit vlastní distribuci linuxu po rozporech v Stampede Linuxu, který pomáhal vyvíjet. Začal vyvíjet distribuci Enoch se zaměřením na rychlost a automatickou instalaci a upgrade balíčků. Po čase se distribuce přejmenovala na Gentoo. Načas pak distribuce ustala, když Robbins přešel na FreeBSD. Po jeho návratu se vývoj obnovil. Daniel Robbins získal časově neomezené právo provozovat obchod s produkty Gentoo. Zisk z něj není používán na rozvoj komunity, ale smí si ho nechat jako náhradu za náklady na rozvoj Gentoo, jež ho údajně stály přes 40 000 amerických dolarů a přivedly do dluhů. Toto rozhodnutí bylo některými členy komunity přijato s rozpaky. V pondělí 26. dubna 2004 Daniel Robbins odstoupil z místa hlavního architekta projektu. Předtím zřídil nadaci Gentoo Foundation a převedl na ni všechna autorská práva. Robbins jmenoval dozorčí radu, než bude další rok zvolena nová. Nadace je otevřena novým členům. Dále vrátil nadaci i právo na provozování obchodu a značky Gentoo. [2]

2.3 Portage

Gentoo Linux používá systém správy balíčků Portage. Obsahuje přes 25000 balíčků. Instalační závislosti jednotlivých balíčků jsou řešeny automaticky.

Přestože systém se jmenuje Portage pracuje se s ním převážně za pomocí `emerge`, což je nástroj příkazové řádky umožňující instalaci, odinstalaci balíčků, synchronizaci Portage stromu a mnoho dalších funkcí. Je napsán v jazyce Python. Stará se o vyřešení závislostí, stažení zdrojových kódů, jejich překlad, následnou instalaci a aktualizaci konfiguračních souborů. Kompilace se provádí v pomocném adresáři a po jejím úspěšném dokončení jsou soubory začleněny do systému.

Kompilace ze zdrojovůch kódů však není podmínkou. Některé programy mohou být nainstalovány z binárních souborů. V tomto případě však z pochopitelných důvodů není optimalizován pro cílovou architekturu.

Emerge má i své nevýhody mezi než patří hlavně nemožnost automaticky odinstalovat již nepotřebné závislosti. Některé tyto nedostatky řeší `Paludis` [14] což je alternativa k `emerge`. Umožňuje například aktualizaci stromu přes SVN, CSV, Git apod., je rychlejší, důsledněji kontroluje závislosti a má komfortnější vyhledávání závislostí.

2.3.1 Struktura portage stromu, organizace dat

Celý strom Portage je vlastně hierarchicky uspořádaná adresářová struktura na disku (viz obr. 2.1) obsahující soubory s ebuildy (viz 2.3.2), soubory s kontrolními součty (`Manifest`, `digest`) a další jako třeba patche. Neměly by se zde vyskytovat velké či binární soubory. Pro ty se používá adresář distfiles, kam jsou také stahovány archivy se zdrojovými kódy a různé další soubory potřebné pro instalaci. Pro ukládání souborů s licencemi se používá adresář `licenses`. Celý strom je většinou umístěn na disku v adresáři `/usr/portage`. Všechny tyto parametry však lze konfigurací změnit.

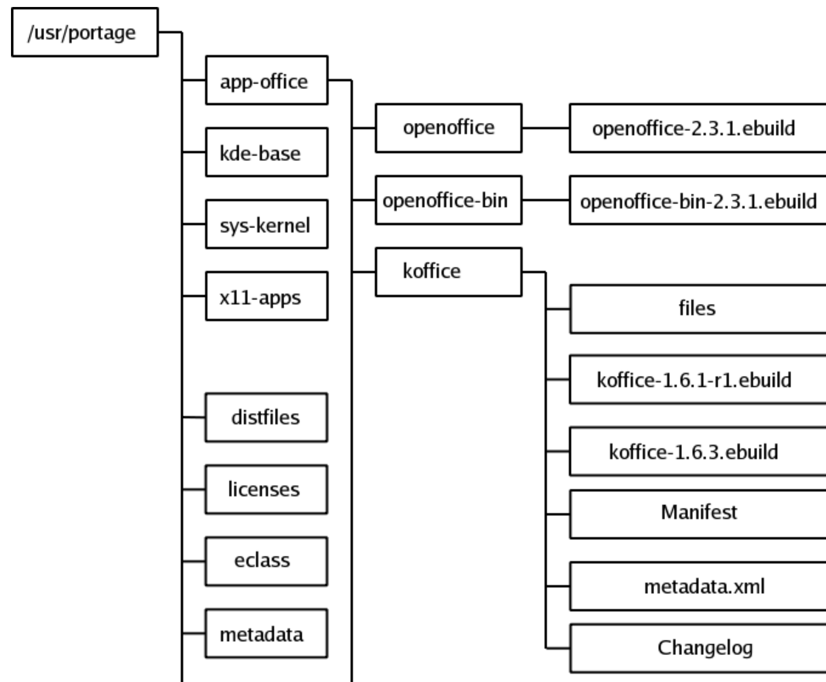
Na kořen stromu jsou napojeny adresáře skupin programů, které sdružují programy s podobným zaměřením a funkcemi. Například ve skupině `sys-kernel` jsou jádra pro různé hardwarové architektury a software pro práci s nimi, ve skupině `app-office` kancelářský software apod. . V těchto skupinách se dále nacházejí adresáře programů a teprve v nich soubory s ebuildy. Protože program může být k dispozici ve více verzích musí mít každá svůj vlastní ebuild. V tomto adresáři se mohou také vyskytovat další soubory a podadresáře potřebné při instalaci programu (patche s podadresáři files apod.).

Celý strom je nutné jednou za čas synchronizovat a aktualizovat tak balíčky. Slouží k tomu příkaz `emerge --sync`.

2.3.2 Ebuild

Ebuild je textový soubor s příponou `.ebuild` uložený v adresáři programu, který obsahuje informace o tom, kde stáhnout zdrojové kódy, jak je přeložit a nainstalovat program, informace o domovské stránce, USE proměnných (viz kapitola 2.3.3), klíčová slova (viz kapitola 2.3.4), ze kterých eclass dědí (viz kapitola 2.3.5) a další. Jde vlastně o shell skript který má danou strukturu.

Ebuild může také popisovat instalaci binárního balíčku. Gentoo používá také formát `.tbz2` pro binární balíčky, což je tar archiv s kompresí bzip2 ve kterém jsou uložena další metadata. Balíček tak může být sestaven na jednom počítači a následně rychle nainstalován na jiném. Tato možnost je výhodná obvzláště v případě serverů.



Obrázek 2.1: struktura Portage stromu

2.3.3 USE proměnné

Další velkou výhodou Portage a kompilace ze zdrojových kódů obecně je možnost zkompilovat program pouze s těmi parametry a funkcemi, které uživatel potřebuje.

Ebuildy mohou mít žádný či více tzv. USE proměnných (USE flags)[10], při jejichž nastavení je program zkompilován s některými dalšími funkcemi. Například USE proměnná `gtk` zapíná podporu grafického uživatelského rozhraní programu v knihovně GTK. Funguje to tak, že třeba u programů vyžívajících při instalaci configure skript přídá parametr `./configure --with-feature gtk`. Uživatel si tak může instalovat programy pouze s funkcemi, které opravdu využije.

Výchozí hodnoty USE proměnných jsou dány používaným profilem[11] a uživatel je může změnit v souboru `/etc/portage/package.use`.

2.3.4 Maskování a klíčová slova

Maskování je způsob označování vhodnosti balíčků pro daný systém. Důkladně testované balíčky jsou označeny jako stabilní (stable), ty u kterých nejsou známy žadné problémy a třeba jen pro tuto architekturu nejsou dostatečně otestovány jsou označeny jako nestabilní (unstable). Naopak software se kterým jsou problémy je označen klíčovým slovem „Hard masked“. Instalace těchto balíčků není doporučována avšak existuje možnost, že budou správně fungovat.

2.3.5 Eclass

Eclass jsou textové soubory s příponou `.eclass` v adresáři `/eclass`. Obsahují kód, který využívá více ebuildů současně. Některé ebuildy jsou si totiž podobné a aby se nemusel stejný

kód pořád opakovat, je možno z těchto tříd dědit. Eclass je tedy něco jako abstraktní třída v objektově orientovaném programování. Ebuild může dědit i z více eclass. Při modifikaci těchto souborů je nutné postupovat velice opatrně a eclass důkladně otestovat, protože případná chyba může mít dopad na více ebuildů.

2.3.6 Overlays

Overlay je název pro neoficiální strom s dalšími ebuildy, které jsou sice udržovány vývojaři Gentoo, ale jsou distribuovány mimo hlavní strom. Existuje několik důvodů pro jejich používání. V případě, že je modifikován ebuild v hlavním stromu, tyto změny vydrží pouze do příští synchronizace. Ebuildy v neoficiálním stromu jsou proti tomuto v bezpečí a to z něj dělá ideální místo pro vývoj a nehrozí také žádné poškození hlavního stromu. Některé ebuildy jsou do nich umísťovány před tím, něž se objeví v hlavním stromu.

2.3.7 Výhody a nevýhody

Jelikož je celý systém od základu kompilován ze zdrojových kódů, je také optimalizován pro konkrétní hardware. Gentoo obsahuje širokou škálu možností nastavení a je dobře zdokumentován. Software ve stromu je obvykle aktuální. Například balíčky pro nová jádra jsou k dispozici téměř okamžitě po jejich uvolnění.

Nevýhodou překladu ze zdojových kódů je náročnost na výpočetní výkon a doba kompilace. Například kompilace KDE (K Desktop Environment) na slabším stroji může trvat i několik hodin. Kompilace však může být spuštěna na pozadí s nízkou prioritou, takže si toho uživatel při běžné práci ani nevšimne. Gentoo také není vhodné pro začátečníky, protože veškerá nastavení si uživatel musí provést sám.

2.4 Souborový systém

Souborový systém je způsob organizace dat na disku, CD nebo jiném médiu. Může být umístěn na serveru a přístupný pomocí síťového připojení nebo může pouze vytvářet abstrakci nad virtuálními daty. Obvykle lze za pomocí adresářů vytvářet stromovou strukturu ve které jsou uloženy soubory s daty. Soubory i adresáře lze podle určitých pravidel daných pro konkrétní souborový systém pojmenovat a přistupovat k nim podle jednoznačné cesty.

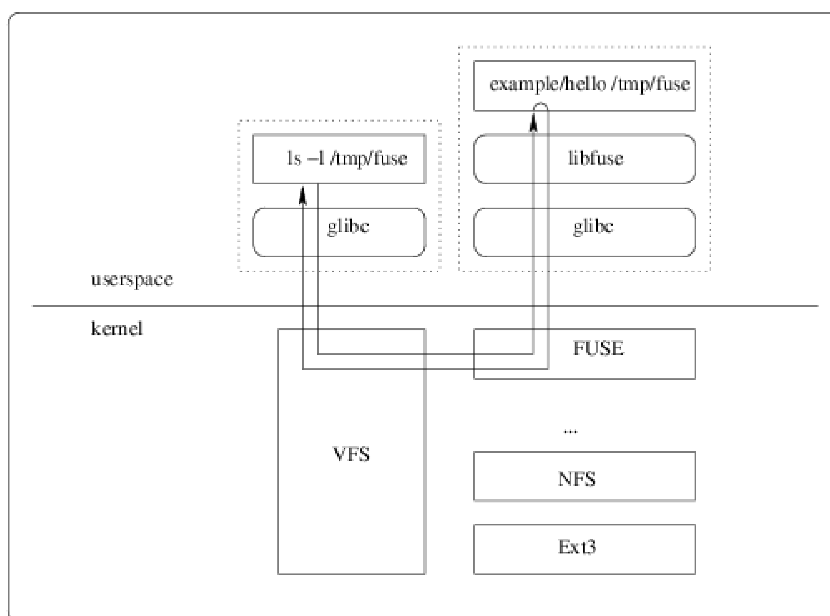
2.4.1 FUSE (Filesystem in Userspace)

FUSE[8] je modul v jádře, který poskytuje propojení k jeho rozhraní (viz obr. 2.2). Pomocí knihovny FUSE je možné vytvořit plně funkční souborový systém který dokáže zpřístupnit téměř jakákoli data jako soubory a adresáře na disku v uživatelském prostoru jádra bez nutnosti psát pro něj zdrojový kód. Má relativně jednoduché API. Implementace FUSE je velice efektivní. Modul a knihovna spolu komunikují přes speciální deskriptor, který je získán otevřením souboru `/dev/fuse`. Tento soubor může být v jednu chvíli otevřen vícekrát.

Na FUSE je již založeno mnoho souborových systémů, například SSHFS, FTPFS, GMailFS.

Vznikl oddělením od projektu A Virtual File System (AVFS), kterého byl původně součástí.

FUSE je dostupný pro většinu unixových systémů (Linux, FreeBSD, NetBSD, OpenSolaris a Mac OS X). Pro linoxová jádra 2.4 a 2.6 (od verze 2.6.14 je součástí jádra).



Obrázek 2.2: struktura Portage stromu

2.5 Relační databáze

Pod pojmem databáze rozumíme soubor dat s definovanou strukturou, která umožňuje vkládání, uchovávání a vyhledávání informací v ní uložených. V dnešní době rozlišujeme širokou škálu databází, od jednoduchých tabulek až k velmi složitým databázím s mnoha milióny záznamů vyžadujícím kvalitní hardwarové zázemí. S počátky databází jsou spojena jména jako Codd či Charles Bachman. Již v roce 1890 vzniká na objednávku státních úřadů v USA první automat na bázi děrných štítků. U jeho zrodu stál Herman Hollerith, jehož firma při fůzi několika firem dává vzniknout IBM. Ta stojí v popředí i v roce 1935, kdy vzniká první digitální počítač pro komerční využití UNIVAC I. V roce 1960 vzniká předchůdce dnešních databázových jazyků COBOL. V 60. letech zakládá Charles Bachman spolu s dalšími výzkumníky seskupení Codasyl, které publikuje základní specifikaci pro programovací jazyky, především pro COBOL. Většina Codasyl kompatibilních databází byla postavena na síťovém modelu, zatímco firma IBM se vydala cestou hierarchického modelu. V roce 1970 přichází Ted Codd s novým návrhem datového modelu, relačním modelem. Dle relační teorie lze pomocí základních operací (sjednocení, kartézský součin, rozdíl, selekce, projekce a spojení) uskutečnit veškeré operace s daty a ostatní operace jsou již jen kombinacemi těchto pěti. Zavádí tedy použití relačního kalkulu a algebry. Databáze mají být nezávislé na fyzickém uložení dat i na použitém jazyce. Pod tlakem událostí se do projektu vkládá i IBM, která přichází s jazykem SQL. První SQL databází byl v roce 1980 Oracle pro počítače VAX. Druhá v řadě přichází i firma IBM s produktem DB2. Osmdesátá léta lze považovat za zlatý věk databází. Proč využíváme databáze? Zjednodušeně řečeno je databáze úložiště dat, které nám umožňuje rychlejší a přímý přístup k datům. V databázových systémech je umožněn paralelní přístup více uživatelů s možností ošetření přístupových práv. A v neposlední řadě je to možnost získávat množiny dat vyhovující námi

zadaným kriteriím.[5]

2.5.1 Jazyk SQL

V 70. letech 20. století probíhal ve firmě IBM výzkum relačních databází. Bylo nutné vytvořit sadu příkazů pro ovládání těchto databází. Vznikl tak jazyk SEQUEL (Structured English Query Language). Cílem bylo vytvořit jazyk, ve kterém by se příkazy tvořily syntakticky co nejblíže přirozenému jazyku (angličtině). K vývoji jazyka se přidaly další firmy. V r. 1979 uvedla na trh firma Relational Software, Inc. (dnešní Oracle Corporation) svoji relační databázovou platformu Oracle Database. IBM uvedla v roce 1981 nový systém SQL/DS a v roce 1983 systém DB2. Dalšími systémy byly např. Progres, Informix a SyBase. Ve všech těchto systémech se používala varianta jazyka SEQUEL, který byl přejmenován na SQL. Relační databáze byly stále významnější, a bylo nutné jejich jazyk standardizovat. Americký institut ANSI původně chtěl vydat jako standard zcela nový jazyk RDL. SQL se však prosadil jako de facto standard a ANSI založil nový standard na tomto jazyku. Tento standard bývá označován jako SQL-86 podle roku, kdy byl přijat. V dalších letech se ukázalo, že SQL-86 obsahuje některé nedostatky a naopak v něm nejsou obsaženy některé důležité prvky týkající se hlavně integrity databáze. V roce 1992 byl proto přijat nový standard SQL-92 (někdy se uvádí jen SQL2). Zatím nejnovějším standardem je SQL3 (SQL-99), který reaguje na potřeby nejmodernějších databází s objektovými prvky. Standardy podporuje prakticky každá relační databáze, ale obvykle nejsou implementovány vždy všechny požadavky normy. A naopak, každá z nich obsahuje prvky a konstrukce, které nejsou ve standardech obsaženy. Přenositelnost SQL dotazů mezi jednotlivými databázemi je proto omezená.[6]

2.5.2 PostgreSQL

PostgreSQL[16] je plnohodnotným relačním databázovým systémem s otevřeným zdrojovým kódem. Má za sebou více než patnáct let aktivního vývoje a má vynikající pověst pro svou spolehlivost a bezpečnost. Běží na všech rozšířených operačních systémech včetně Linuxu, UNIXů (AIX, BSD, HP-UX, SGI-IRIX, Mac OS X, Solaris, Tru64) a Windows. Stoprocentně splňuje podmínky ACID, plně podporuje cizí klíče , operace JOIN, pohledy, spouště a uložené procedury. Obsahuje většinu SQL92 a SQL99 datových typů, např. INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL a TIMESTAMP. K systému existuje kvalitní volně dostupná dokumentace včetně českých překladů FAQ a FAQ pro o.s. fy. Microsoft. PostgreSQL je šířen pod BSD licencí, která je nejliberálnější ze všech open source licencí. Tato licence umožňuje neomezené používání, modifikaci a distribuci PostgreSQL. PostgreSQL je možno šířit se zdrojovými kódy nebo bez nich, zdarma nebo komerčně. PostgreSQL umožňuje běh uložených procedur napsaných v několika programovacích jazycích, v Perlu, v Pyhon, v jazyku C nebo v speciálním PL/pgSQL - jazyku vycházejícím z PL/SQL fy. Oracle. Existují PostgreSQL varianty JDBC, ODBC, dbExpress, Open Office, PHP, .NET Perl nativních rozhraní. K PostgreSQL existuje překladač Embedded SQL pro C a C++. Předností systému PostgreSQL je rozšiřitelnost. Systém může být bezproblémově rozšiřován o nové datové typy, funkce operátory, agregační funkce, procedurální jazyky. Díky tomu mohly vzniknout následující rozšíření: PostGIS - podpora pro geografické informační systémy, TSearch2- podpora fulltextového vyhledávání , Slony-I - master to multiple slaves replikace. Na serveru pgfoundry[15] je k dispozici několik desítek doplňků včetně doplňků rozšiřující o funkcionalitu MySQL, SQL Serveru a Oraclu.[4]

Kapitola 3

Praktická část

3.1 Analýza

3.1.1 Požadavky na program

Od programu je požadováno, aby dokázal zpřístupnit data v relační databázi jako klasické úložiště Gentoo Portage za využití knihovny FUSE. Musí tedy nejen zobrazit správně data, ale musí být schopen provádět veškéré další operace se soubory a adresáři jako by šlo o klasický souborový systém. Bude nutné implementovat funkce jako čtení, mazání, zapisování do souborů, vytváření adresářů a převést tyto události v operace nad relační databází. Musí být také ale dostatečně rychlé protože se pracuje s velkým množstvím malých souborů a právě rychlost vyhledávání informací a počítání zavislostí ebuildů ve stromu bývá časově náročná. Zde by mohlo být použití relační databáze, která je právě pro takový učel navržena, výhodou.

Bude také nutné dobře zvolit implementační jazyk. Podle povahy dat a požadovaných funkcí bude jistě výhodou využití objektového přístupu.

3.1.2 Ukládaná data

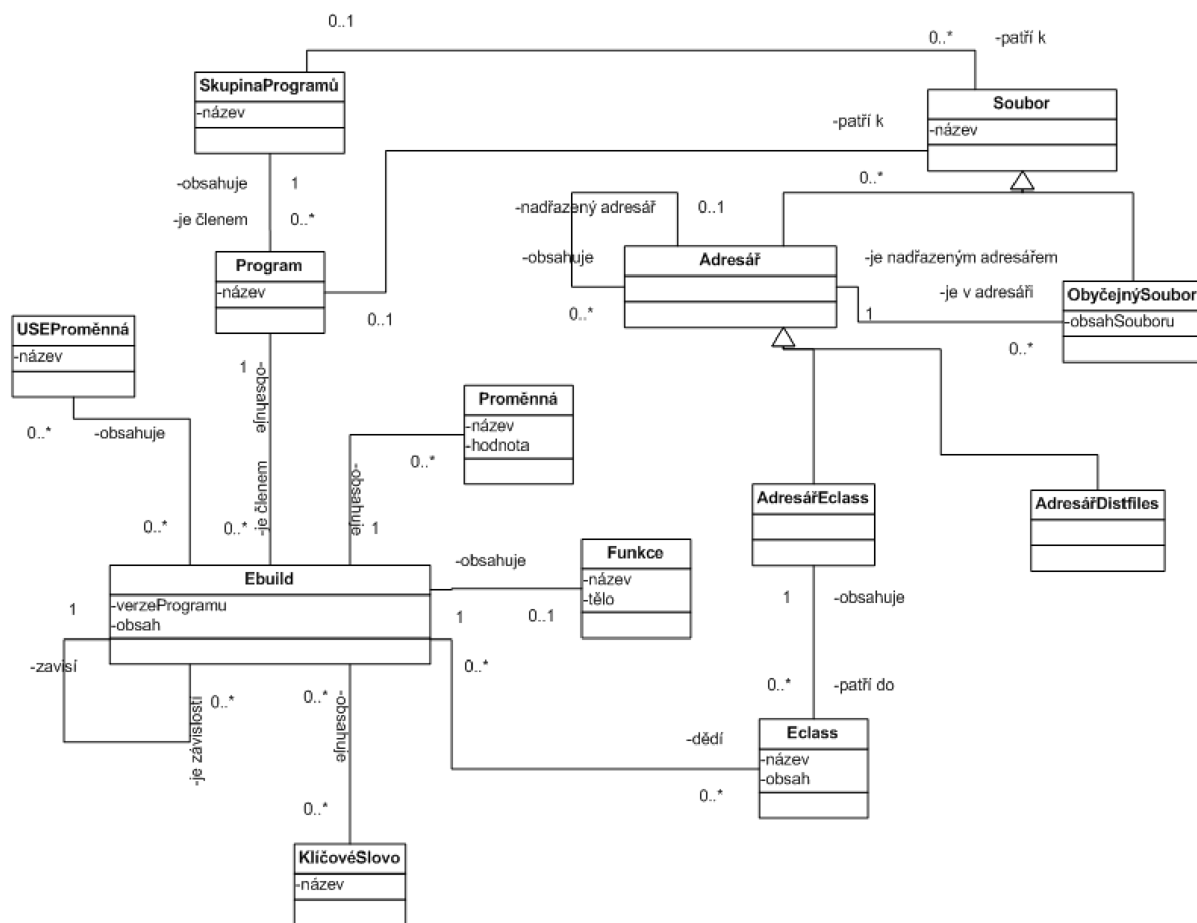
Uložení dat v databázi je trochu odlišné od ukládání dat v souborovém systému. Bude nutné vytvořit tabulky a do nich uložit ebuildy, eclass, skupiny programů, jednotlivé programy, metadata, digest, manifest a další soubory jako jsou patche a podobně.

Některé soubory, které mají pevně danou strukturu by bylo vhodné zparsovat a uložit do více tabulek. Přímo se nabízí soubory s ebuildy. Soubory s eclass mohou být uloženy v původní podobě. Dále bude třeba ukládat do jednotlivých tabulek skupiny programů, programy a další soubory a adresáře, které jsou třeba při instalaci balíčků, ale nemusí se vyskytovat u všech programů (obsah adresáře files apod.).

Diagram tříd, který znázorňuje data, která bude třeba uložit do databáze je znázorněn na obrázku 3.1.

3.1.3 Způsob uložení ebuildů v databázi

U ebuildů by mohlo být přínosné ukládat některé informace do více databázových tabulek a umožnit tak vyhledávání podle určitých parametrů. Těmito informacemi (viz obr. 3.1) jsou závislosti, eclass ze kterých dědí , USE proměnné, klíčová slova, některé další důležité proměnné. Další otázkou je jak bude probíhat zpětné sestavení ebuildu aby bylo korektní a



Obrázek 3.1: Diagram tříd ukládaných dat

nepříliš časově náročné. Problém by mohl nastat při instalaci programu, kdy se také ověřuje kontrolní součet ebuildu. Řešení by mohlo být následující:

1. Ebuild sice zparsovat, ale ještě uložit jeho původní, neupravený obsah zvlášť a ten pak předávat přes FUSE společně s dalšími soubory, které vyžadují digitální podpis. Pak jsou samozřejmě kontrolní součty vpořádku, protože obsah souboru je původní.

2. Ebuild rozparsovat a poznamenavat si další informace tak, aby se dal znovu z jednotlivých částí složit takovým způsobem, že digitální podpis bude stejný s původním před parsováním. To znamená složit do původní podoby jakou měl před parsováním. Zde se zřejmě bude spousta problémů například s bilými znaky.

3. Ebuild rozparsovat a když pak je přes fuse vyžadován jako soubor, tak ho složit a vygenerovat k němu nový podpis. Toto řešení je zřejmě zbytečně náročné na výpočet (neustálé počítání součtů), ale šlo by to urychlit vhodným použitím vyrovnávací paměti nebo předpočítáním součtů při update ebuildu či jeho souboru

4. Ebuild rozparsovat a o podpisy se nestarat. Jednoduše vypnout ověřování podpisu při emerge (například vhodným nastavením v souboru `make.conf`).

Instalační závislosti ebuildu

Ukládat zvlášť závislosti ebuildů by mohlo být užitečné při hromadném odmaskování balíčků, kdy instalovaný program je závislý na více balíčcích, které jsou maskovány některým klíčovým slovem.

Zpracování obsahu proměnných `DEPEND` a `RDEPEND` (v obrázku 3.1 reflexivní asociace u třídy `Ebuild`), které definují závislosti pro kompilaci a běh programu však bude ne zcela triviální problém protože se mohou při popisu vyskytnou výrazy (viz [9]):

1. vnoření

   ```
   DEPEND="!build? (
       gcj? (
           gtk? (
               x11-libs/libXt
               x11-libs/libX11
               x11-libs/libXtst
               x11-proto/xproto
               x11-proto/xextproto
               >=x11-libs/gtk+-2.2
               x11-libs/pango
           )
           >=media-libs/libart_lgpl-2.1
       )
       >=sys-libs/ncurses-5.2-r2
       nls? ( sys-devel/gettext )
   )"
   ```

   Je několik možností jak řešit vnořené závislosti:

   (a) pamatovat si celé obsahy proměnných *DEPEND a vyhodnocovat je až bude třeba a použít algoritmus např. z emerge, equery, apod.

   (b) parsovat *DEPEND proměnné a do databáze je ukládat rozparsované a ukládat je do nějaké hierarchické struktury (stromu)

2. použití proměnných

   ```
   RDEPEND=">=net-libs/ccrtp-1.5.0
       >=dev-cpp/commoncpp2-1.4.2
       $(qt_min_version 3.3.0)
       media-libs/libsndfile
       dev-libs/boost
       speex? ( media-libs/speex )
       ilbc? ( dev-libs/ilbc-rfc3951 )
       zrtp? ( net-libs/libzrtpcpp )
       media-libs/alsa-lib"
   ```

3. logické výrazy

```
DEPEND="|| ( app-misc/foo app-misc/bar )"
DEPEND="baz? ( || ( app-misc/foo app-misc/bar ) )"
DEPEND="use-flag? ( app-misc/foo ) : ( app-misc/bar )"
DEPEND="use-flag?  ( app-misc/foo )
    !use-flag? ( app-misc/bar )"
```

Zde by bylo třeba zavést pravidla pro substituci, která by prevedla výraz na standardní notaci závislostí, případně oznámila chybu při použití neznámé proměnné, ale tady by už byla zřejmě potřeba syntaktické analýza výrazů.

USE proměnné a klíčová slova

Ukládání těchto proměnných by mohlo být užitečné při vyhledávání konkrétních balíčků podle zadaných parametrů (vztah třídy `Ebuild` s třídami `Eclass` a `KlíčovéSlovo` na obr. 3.1).

Proměnné a funkce

U ebuildu by možná mohlo být užitečné uložit některé důležité proměnné a funkce (třídy `Proměnná` a `Funkce` v obr. 3.1).

3.1.4 Další soubory a adresáře

Dále je nutné uložit některé další soubory a adresáře vyskytující se ve stromu. Na obrázku 3.1 třída `Soubor` a její dceřiné třídy `Adresář` a `ObyčejnýSoubor`.

3.1.5 Data v adresáři distfiles

O obsah adresáře distfiles, kde jsou uloženy stažené zdrojové kódy a další soubory potřebné k instalaci balíčku se není třeba starat, protože umístění tohoto adresáře lze změnit nastavním proměnné `DISTDIR` v souboru `make.conf` (viz `man make.conf`).

3.1.6 Velikost souborů ukládaných v databázi

Další otázkou je, jak velké soubory musí být souborový systém schopen uchovávat. Ve stromu portage se nenacházejí žádné velké soubory, proto bylo za pomocí skriptu zjištěn největší soubor ve stromu, přidána rezerva a tato hodnota byla použita jako maximální možná velikost souboru. Konstantu měnící tuto vlastnost lze později změnit a přizpůsobit tak velikost dat novým podmínkám.

3.1.7 Implementované funkce v knihovně FUSE

FUSE se bude starat o volání námi implementovaných funkcí pro různé operace se soubory, které je převedou na operace s databází. Budou to tyto funkce:

1. `getattr` – zjištění informací o souboru (typ, velikost, přístupová práva...), toto bude nejpoužívanější funkce, měla by být implementována co nejefektivněji v ideálním případě použít i vyrovnávací pamět
2. `readdir` – čtení obsahu adresáře

3. `open` – otevření souboru
4. `read` – čtení obsahu souboru
5. `mknod` – vytvoření souboru
6. `mkdir` – vytvoření adresáře
7. `unlink` – zrušení souboru
8. `mkdir` – zrušení adresáře
9. `rename` – přesun souboru
10. `truncate` – zkrácení souboru
11. `utimens` – změn času modifikace souboru
12. `write` – zápis do souboru
13. `statfs` – zjištění informací o souborovém systému

3.1.8 Přístupová práva k souborům

Přístupová práva nebude třeba ukládat, protože všechny soubory je budou mít stejné. K souborům portage stromu budou mít právo zápisu pouze privilegovaní uživatelé (root) a ostatní pouze právo čtení.

3.2 Návrh

3.2.1 Výběr programovacího jazyka

Původně jsem chtěl použít jazyk Python, který by byl vhodný zejména z důvodu rychlejšího vývoje programu a široké škále funkcí pro práci s textem, ale nakonec jsem se rozhodl pro C/C++. V pythonu jsem měl problémy s použitím knihovny pro FUSE. Interpret oznamoval chybu v jedné funkci knihovny. Kromě toho žádná verze není v Portage označena jako stabilní. Verze pro C/C++ funguje bez problémů. Navíc jazyk C++ má tu výhodu, že jde o kompilovaný jazyk, tudíž bude při dobré implementaci jistě rychlejší než interpretovaný, skriptovací jazyk Python.

3.2.2 Výběr databáze

Jako databáze byla vybrána PostgreSQL. Zejména díky podpoře mnoha pokročilých vlastností a své volné licenci (viz 2.5.2).

Pro jazyk C++ a databázi PostgreSQL jsou k dispozici tyto tři hlavní knihovny:

- libpq – knihovna pro jazyk C
- libpqpp – wrapper libpq pro jazyk C++, nepodporuje výjimky
- libpqxx – další knihovna pro C++

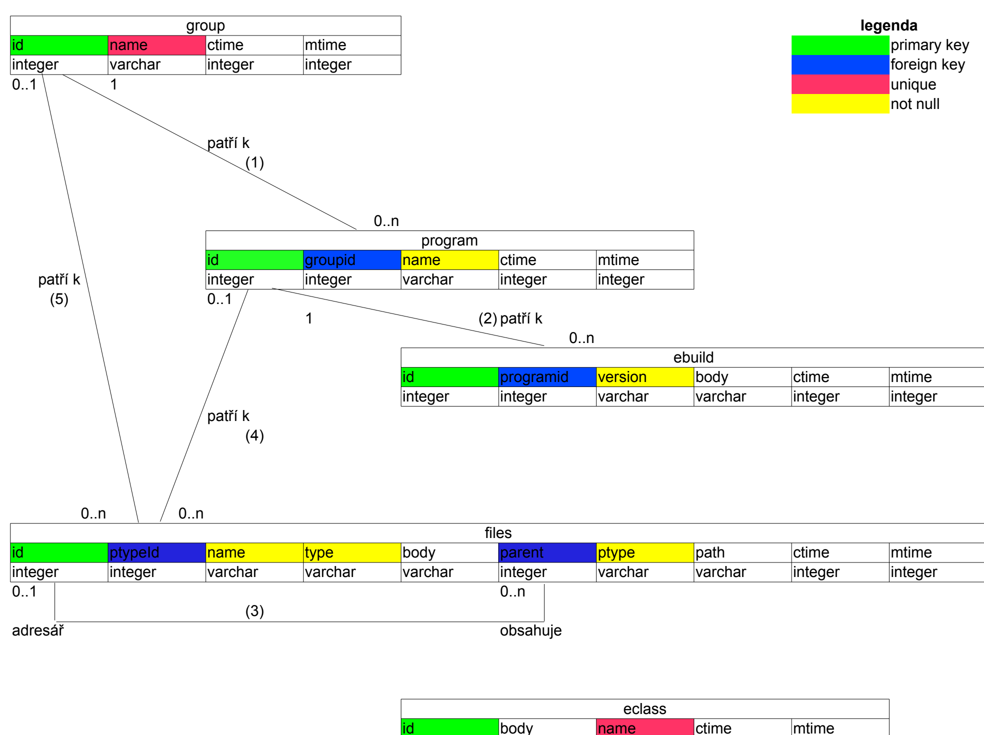
Nakonec byla vybrána libpqxx, protože jde o plnohodnotnou knihovnu s podporou výjimek a ne o pouhý wrapper knihovny jazyka C. Existují ještě další knihovny, které třeba vytvářejí abstrakci nad více databázemi, ale o nich není k dispozici takové množství podkladů jako těchto výše zmíněných.

3.2.3 Návrh databáze

Ze všech možností jak uloži ebuild se jako nejideálnější vzhledem k náročnosti na výkon i implementaci jeví ukládání se zparsovanými daty i původní obsah souboru. Není třeba tedy přepočítávat kontrolní součty, složitě skládat data do původní podoby a ani vypínat tyto mechanismy, což se určitě příznivě projeví i na výkonu programu. Další výhodou tohoto přístupu je fatk, že program může být navržen tak, aby mohlo být implementačně náročné parsování ebuildů přidáno až dodatečně, protože na zobrazování dat v souborovém systému nebude mít přímý vliv.

Databázovým datovým typem pro uchování obsahu souboru byl zvolen varchar poněvadž ve stromu jsou uchovávány v drtivé většině textové soubory.

Z diagramu tříd 3.1 byly vytvořeny následující databázové tabulky:



Obrázek 3.2: Návrh databázových tabulek

Tabulka group

Tato tabulka reprezentuje třídu `SkupinaProgramů` z obrázku 3.1. Záznamy v tabulce group budou adresáře skupin. Tabulka skupiny bude mít pouze dva atributy a to identifikátor skupiny a název skupiny.

Tabulka program

Tato tabulka reprezentuje třídu `Program` z obrázku 3.1. Bude obsahovat kromě identifokátoru a názvu programu také atribut groupid, která bude vyjadřovat příslušnost do skupiny – obr. 3.2 vazba 1. Každý program náleží právě do jedné skupiny a každá skupina může mít žádný nebo více programů.

Tabulka ebuild

Tato tabulka reprezentuje třídu `Ebuild` z obrázku 3.1. Parsování ebuildů se ukázalo jako docela náročné, a proto se prozatím do tabulky bude ukládat pouze identifikátor, verze programu (není třeba si pamatovat celý název souboru, protože je dán názvem programu + verze + přípona ebuild), obsah souboru s ebuildem a atribut programid, která vyjadřuje příslušnost k programu – obr. 3.2 vazba 2. Program může mít i více ebuildů. Pro každou verzi nebo revizi té stejné verze jeden .

Tabulka eclass

Tabulka eclass slouží k ukládání souboru s eclass (viz třída eclass v obrázku 3.1). Každý záznam má svůj identifikátor, název a obsah souboru s eclass.

Tabulka files

Tato tabulka slouží k ukládání všech ostatních souborů a adresářů. Jsou to třídy `Adresář` a `ObyčejnýSoubor` z obrázku diagramu tříd 3.1. Atribut id je identifikátor. ParentDir odkazuje na nadřezený adresář – obr. 3.2 vazba 3. Pokud má hodnotu NULL, jedná se o kořenová adresář. V atributu path je uložena cesta k souboru, name je název souboru. Do této tabulky budou ukládány zároveň soubory i adresáře, a proto je v atributu ftype uložena textová informace o tom, zdali se jedná o adresář nebo obyčejný soubor. V tabulce se může vyskytovat více kořenových adresářů, protože každá skupina programů a každý program bude mít svůj vlastní kořenový adresář. Snadno se tak pozná, které soubory patří ke kterým programům, což může urychlit vyhledávání těchto souborů. Položka ptype bude vyjadřovat příslušnost souboru k určitému elementu portage stromu. Těmito elementy mohou být skupiny programů, program, kořen Portage stromu nebo adresář s eclass (možné hodnoty tedy budou GROUP, PROGRAM, ECLASS, ROOT). Atribut pTypeId je identifikuje konkrétní element stromu ke kterému soubor patří – obr. 3.2 vazby 4 a 5. Ve sloupci body je uložen obsah souboru nebo NULL pokud se jedná o adresář.

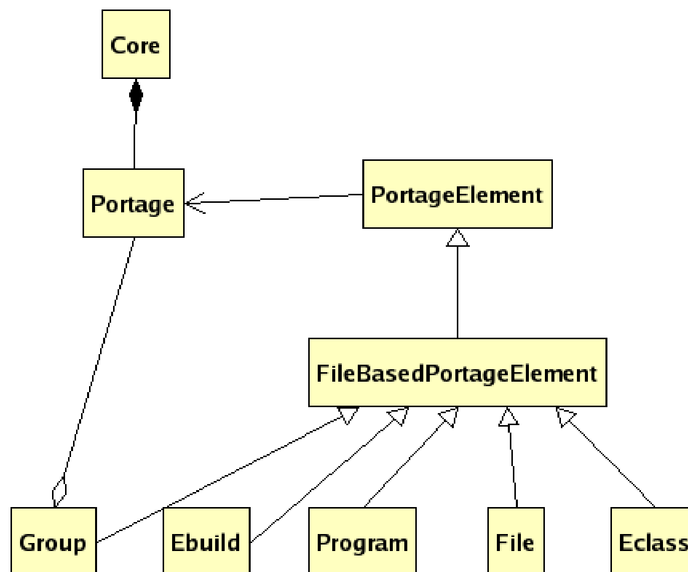
Společné atributy tabulek

Všechny tabulky mají parametry ctime ve kterém je uloženo datum a čas vytvoření souboru a mtime kde je uloženo datum a čas poslední modifikace souboru. Tyto informace jsou nutné při synchronizaci některých overlay, kde se zjišťuje aktuálnost souboru podle data poslední modifikace.

V tabulkách skupiny a programu by bylo možné mít ještě položky, ve kterých by byli uloženy obsahy souborů `metadata.xml`, `Changelog` a `Manifest`, ale práce s těmito soubory by bylo zbytečně implementačně náročné, proto budou také uloženy v tabulce files.

3.2.4 Návrh programu

Za pomocí programu Umbrello UML modelátor byl vytvořen objektový návrh programu (viz obrázek 3.3). Byly navrženy následující třídy a moduly.



Obrázek 3.3: Návrh programu – diagram hlavních tříd

Modul Core

Vstup do programu (funkci main) obsahuje modul Core (viz. obr. 3.4). Zde budou také definovány všechny funkce se souborovým systémem, které bude volat FUSE. Ve funkci main se objevuje pouze jediný objektu, který obsahuje metody pro práci se souborovým systémem. Tím objektem je jediná instance třídy Portage.

Třída Portage

Převádí příkazy souborového systému na vytváření objektů, které reprezentují jednotlivé elementy stromu Portage a volání jejich metod. Tato třída (viz. obr. 3.5) vytváří abstrakci nad celou databází a obsahuje metody:

- write – zapíše data do souboru daného cestou
- removeDir – zruší adresář
- removeFile – zruší soubor
- createDir – vytvoří adresář
- createFile – vytvoří soubor
- getFileInfo – vrátí informace o souboru
- rename – přejmenování a přesun souborů a adresářů

```
Core
- operations : struct fuse_operations = {.getattr = getAttribs,.readdir = readDir,.open = open,.read = read,};
- portage : Portage*
- logger : undef
+ init()
+ main(argc : int, argv : char**) : int
+ getattribs(path : const char*, stbuf : struct stat*) : int
+ readDir(path : const char*, buf : void*, filler : fuse_fill_dir_t, offset : off_t, fi : struct fuse_file_info*) : int
+ open(path : const char*, fi : const char*) : int
+ read(path : const char*, buf : char*, size : size_t, offset : off_t, fi : struct fuse_file_info*) : int
+ access(path : const char*, mask : int) : int
+ readlink(path : const char*, buf : char*, size : size_t) : int
+ mknod(path : const char*, mode : mode_t, r_dev : dev_t) : int
+ mkdir(path : const char*, mode : mode_t) : int
+ symlink(from : const char*, to : const char*) : int
+ unlink(path : const char*) : int
+ rmdir(path : const char*) : int
+ rename(from : const char*, to : const char*) : int
+ link(from : const char*, to : const char*) : int
+ chmod(path : const char*, mode : mode_t) : int
+ chown(path : const char*, uid : uid_t, gid : gid_t) : int
+ truncate(path : const char*, size : off_t) : int
+ utimens(path : const char*, ts : const struct timespec) : int
+ write(path : const char*, buf : const char*, size : size_t, offset : off_t, fi : struct fuse_file_info*) : int
+ statfs(path : const char*, stbuf : struct statvfs*) : int
+ release(path : const char*, fi : struct fuse_file_info*) : int
+ fsync(path : const char*, isdatasync : int, fi : struct fuse_file_info*) : int
+ setxattr(path : const char*, name : const char*, value : const char*, size : size_t, flags : int) : int
+ getxattr(path : const char*, name : const char*, value : const char*, size : size_t) : int
+ listxattr(path : const char*, list : char*, size : size_t) : int
+ removexattr(path : const char*, name : const char*) : int
+ linkedDirTransform(portageDirPath : string) : string
```

Obrázek 3.4: Návrh modulu Core

- getElement – nejdůležitější metoda, podle zadané cesty vytvoří objekt, pokud tedy cesta vede k souboru s ebuildem, tak vytvoří objekt reprezentující ebuild apod., vrací obecný objekt PortageElement, který je možno přetypovat a dále s ním pracovat

Třída PortageElement

Obecný objekt ve stromu Portage je reprezentován třídou PortageElement (viz. obr. 3.6). Každý objekt má svůj identifikátor, který jej jednoznačně odlišuje od ostatních v databázové tabulce. Obsahuje také ukazatel na instanci objektu Portage ke kterému náleží aby mohl využívat společné zdroje. Metody třídy:

- write – zapíše obsah objektu do databázové tabulky
- remove – odstraní objekt z databáze
- create – vytvoří záznam v databázi

Všechny tyto metody fungují tak, že se nejdříve naplní atributy objektu, ,je zavolána metoda, která z těchto atributů vygeneruje SQL kód, který se následně provede. A v případě jakékoli chyby je vyvolána patřičná výjimka (viz 3.2.4).

Portage

- groups : map<string, Group*>
- pool : stack<connection*>
- cache : Cache
- eclassEnable : bool
- distfilesEnable : bool

+ init()
+ createFile(path : string, body : string)
+ readFile(path : string) : string
+ deleteFile(path : string)
+ createDir(path : string)
+ deleteDir(path : string)
+ readDir(path : string) : vector<string>
- parsePath(path : string) : PortageElement
+ getElement(path : string, element : PortageElement*) : enum elementTypes
+ getFileType(path : string) : enum fileTypes
+ loadGroups()
+ fileInfo(path : string) : enum fileTypes
+ write(path : string, content : string)
+ writeContentToString(content : string, new_data : const char*, offset : off_t, size : size_t)
+ releaseConn(conn : connection*)
+ rename(from : string, to : string)

Obrázek 3.5: Návrh třídy Portage

Třída FileBasedPortageElement

Rozšiřuje třídu PortageElement. Jde o rodičovskou třídu pro všechny elementy, které jsou zobrazovány ve formě souborů. Obsahuje navíc atributy ctime pro zaznamenání času vytvoření souboru a mtime pro zaznamenéní poslední modifikace souboru (viz. obr. 3.7). Pokud se v budoucnu bude zaznamenávat více informací o souborech, pak právě tato třída bude rozšiřována o další atributy (například přístupová práva). Obsahuje zatím pouze metodu:

- updateTime – zapíše do databáze změny časů vytvoření souboru a poslední modifikace

Třída Group

Reprezentuje skupinu programů (viz. obr. 3.8). Skupin programů je v Portage něco přes 150, v hierarchii stromu Portage jsou nejvýše a při každém přístupu k nižším elementům bude nutné je načítat z databáze, proto by bylo nejlepší ukládat tyto objekty také do vyrovnávací paměti. Postačí asociativní pole `map`, kde klíčem bude název skupiny. Nutné bude zajistit aktualizaci při každé změně skupiny. Rozšiřuje třídu FileBasedPortageElement pouze o jediný atribut a tím je název skupiny a o metody:

- getProgramNames, getFileNames – vracející názvy programů ve skupině resp. názvy dalších souborů
- getProgram – vrátí objekt programu ve skupině podle názvu
- createProgram – vytvoří program s daným názvem
- updateCache – aktualizuje vyrovnávací paměť objektu Portage

PortageElement
- id : int = 0
- portage : Portage*
+ PortageElement(portage : Portage*)
+ Portage(portage : Portage*, id : int)
+ create()
+ createIfNotExist()
+ createForce()
+ update()
+ remove()
+ exist() : bool
+ load()
+ loadById(portage : Portage*, id : int) : PortageElement
+ count() : int
+ setLgr()
+ updateCache()
+ write(content : string)

Obrázek 3.6: Návrh třídy PortageElement

FileBasedPortageElement
- mtime : unsigned int
- ctime : unsigned int
+ updateTime()
+ updateTime(tableName : string)

Obrázek 3.7: Návrh třídy FileBasedPortageElement

Třída Program

Reprezentuje adresář programu (viz. obr. 3.9), ve kterém jsou soubory s ebuildy. Rozšiřuje třídu FileBasedPortageElement pouze o jediný atribut a tím je název programu a o metody:

- getEbuildNames, getFileNames – vracející názvy programů ve skupině resp. názvy dalších souborů
- getEbuild – vrátí objekt programu ve skupině podle názvu
- createEbuild – vytvoří program s daným názvem

Třída Ebuild

Reprezentuje ebuild programu (viz. obr. 3.10). Rozšiřuje třídu FileBasedPortageElement o atribut ve kterém je uložena verze programu a atribut s obsahem souboru. V budoucnu bude tento objekt obsahovat mnohem více atributů, kterými bodou klíčová slova, USE proměnné, závislosti atp. .

Třída File

Tato třída reprezentuje obecný soubor nebo adresář v Portage stromu (viz. obr. 3.11). Rozšiřuje třídu FileBasedPortageElement o atributy parent (identifikátor nadřazeného adresáře), ptype (typ elementu ke patří - může to být kořen stromu, program , nebo skupina

```
Group
- name : string
- metadataxml : string
+ getProgramNames() : vector<string>
+ getProgram(name : string) : Program
+ updateGroupCache()
+ getOwnFileNames(fileNames : vector<string>)
+ getFile(path : string) : ProgramFile*
+ getFileNames(fileNames : vector<string>)
+ createProgram(name : string)
```

Obrázek 3.8: Návrh třídy Group

```
Program
- groupId : int
- name : string
- changelog : string
- manifest : string
- metadataxml : string
+ getEbuildNames() : vector<string>
+ getEbuild(version : string) : Ebuild
+ getEbuildVersions() : vector<string>
+ getEbuildVersion(ebuildFileName : string) : string
+ getProgramFile(fileName : string) : ProgramFile*
+ getProgramFileNames() : vector<string>
+ getProgramOwnFileNames(fileNames : vector<string>)
+ createEbuild(name : string)
```

Obrázek 3.9: Návrh třídy Program

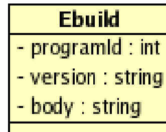
programů), ptypeId (identifikátor tohoto elementu), path (absolutní cesta k souboru), name (jméno souboru), type (typ souboru - adresář/soubor) a body (obsah souboru). Obsahuje další metody:

- getRoot – vrací objekt reprezentující kořen
- getFile – vrácí objekt souboru daný cestou
- createSubFile – vytvoří podadresář nebo obyčejný soubor, může vytvářet jen objekt, který je adresářem

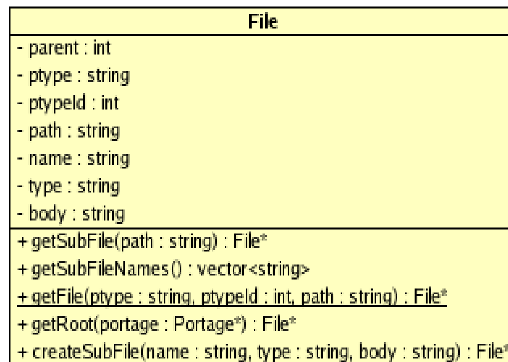
Třída Eclass

Reprezentuje soubor s eclass (viz. obr. 3.12). Rozšiřuje třídu FileBasedPortageElement o atribut ve kterém je uložena verze programu a atribut s obsahem souboru. Obsahuje metody:

- getEclass – podle názvu vrátí příslušný objekt Eclass
- getEclassFileNames – vrátí názvy všech eclass



Obrázek 3.10: Návrh třídy Ebuild



Obrázek 3.11: Návrh třídy File

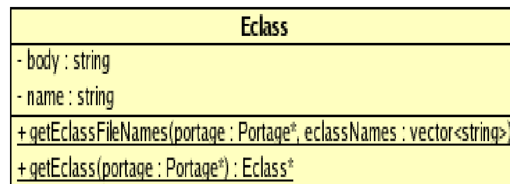
Výjimky

K ošetření různých chybových stavů, které mohou nastat je třeba vytvořit několik výjimek.

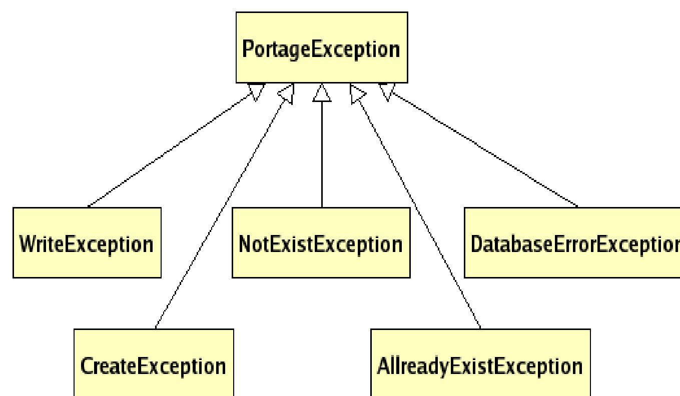
- PortageException – Obecná výjimka. Dědí z výjimky runtime_error. Všechny definované další výjimky z ní budou dědit.
- NotExistException – Výjimka je vyvolána metodami při požadavku na neexistující objekt v portage (například při získávání obsahu adresáře který neexistuje).
- AllreadyExistException – Výjimka je vyvolána při pokusu vytvořit objekt, který již existuje (například při pokusu vytvořit eclass když už jedna se stejným jménem existuje).
- CreateException – Vyjímka je vyvolána při pokusu vložit do databáze nekorektní data (porušení integritních omezení apod.).
- DatabaseErrorException – Vyjímka je vyvolána pokud vznikne chyba při komunikaci s databází (například proto, že databázový stroj není spuštěn nebo databáze neexistuje).
- WriteException – Výjimka vzniká při pokusu změnit atributy souboru (datum vytvoření a poslední modifikace) na nekorektní hodnoty nebo chybném zápisu do souboru (špatně nastavený offset apod.).

3.3 Implementace

Objekty byli naprogramovány podle návrhu avšak museli být provedeny určité změny.



Obrázek 3.12: Návrh třídy Eclass



Obrázek 3.13: Hierarchie navržených výjimek

Při generování SQL dotazů jsou použity obecné konstrukce aby bylo možno program používat s více databázemi. Kód specifický pro každou databázi je uzavřen do bloků `#ifdef` + `#endif` a při komplicaci se přeloží pouze ta část kódu, která je požadována. Zatím je podporována pouze databáze Postgres.

Dále byl vytvořen modul `utils`, s dalšími pomocnými funkcemi a třída `Constants` ve které jsou definovány všechny používané konstanty, jako jsou názvy databázových tabulek a podobně.

3.3.1 Logování událostí

Při běhu programu dochází k mnoha událostem (čtení souboru, práce s databází, vytváření objektů atd.), při kterých může dojít k chybovým stavům. Tyto stavy je třeba zaznamenat. Zvlášť užitečná je tato vlastnost při ladění. Pro tento účel byla zvolena knihovna log4cxx [13], která podporuje různé úrovně logování. Měnit tyto úrovně je dokonce možné bez rekompilace programu pomocí konfiguračního souboru.

3.3.2 Modul Core

Ṕrí spuštění funkce `main` dojde k inicializaci loggeru, k vytvoření globálního objektu Portage a zaregistrování funkcí struktury `fuse_operations`, které jsou volány modulem v jádře při operacích se soubory přes knihovnu FUSE. Tyto funkce pouze volají metody globálního objektu Portage.

3.3.3 Třída Portage

Při svém vzniku nejdříve inicializije svůj logger, loggery tříd ebuildů, programů a tak dále, vytvoří spojení s databází a načte všechny skupiny programů do vyrovnávací paměti (toto je jediná situace, kdy objekt Portage získává data z databáze přímo a ne pomocí jiného objektu).

Nejdůležitější metodou, kterou bylo třeba implementovat je getElement, která podle svého parametru, kterým je cesta k souboru, vrátí objekt reprezentující tento soubor. Pracuje tak, že si cestu k souboru rozdělí na části (na jednotlivé adresáře podle úrovně zanoření) a postupuje od kořenového adresáře až k požadovanému souboru a při tom vytváří objekty reprezentující jednotlivé „meziadresáře". Každý následující objekt je získán pomocí metody objektu předchozího.

3.3.4 Ostatní třídy

Ostatní třídy byli implementovány dle návrhu a žádné významější změny nemuseli být provedeny (viz programová dokumentace na přiloženém CD).

3.3.5 Vytváření nového adresáře v kořenovém adresáři

Při vytváření nových adresářů v kořenovém adresáři vzniká problém s tím, že nelze jednoznačně určit zdali se jedná o adresář skupiny nebo o adresář jiný a tudíž do které databázové tabulky jej zapsat. Nové skupiny však vznikají jen zřídka a jejich počet je relativně malý (jen asi 150). Proto jsou v množině `GROUP_NAMES` ve třídě Constants uloženy názvy všech možných skupin a při vytváření nováho adresáře je jeho název testován na přítomnost v množině a podle toho zapsán do určité tabulky.

3.3.6 Vytváření nového adresáře v adresáři skupiny

S vytvářením adresářů v adresáři skupiny je situace podobná, s tím rozdílem, že není možné určit natož si pamatovat všechny možné adresáře programů. V adresáři skupiny se však adresáře, které nejsou adresáři programu vyskytují vyjímečně a většinou začínají nějakým speciálním znakem (například pro soubory v overlay synchronizovaných pomocí SVN jsou umístěny adresáře `.svn`). Proto v proměnné `FILE_NAMES_BEGINS` jsou uloženy řetězce, kterými když bude vytvářený adresář začínat, tak bude uložen do databázové tabulky files.

3.3.7 Connection pool

Protože čtecí operace nad souborovým systémem jsou prováděny vícevláknově (typicky u čtení obsahu adresáře) je nutné vytvářet více připojení k databázi aby nedocházelo ke kolizím, protože další transakce může být zahájena až po ukončení předchozí. Vytváření

nových spojení a jejich uzavírání po skončení práce s nimi by bylo nejen značně pomalé, ale také by to neúměrně zatěžovalo databázový server.

Connection pool je implementován jednoduše jako zásobník. Pokud metoda potřebuje připojení k databázi, tak jí je přiděleno z tohoto zásobníku a po ukončení práce jej opět vrátí zpět. Teprve pokud je prázdný, tak se vytvoří nové spojení. Naopak pokud je otevřeno mnoho spojení, která již nejsou potřeba, dojde k jejich postupnému uzavírání. Maximální počet otevřených spojení lze nastavit.

3.3.8 Odkazy na adresáře v souborovém systému

Přestože adresáře jako distfiles nemuseli být v programu uvažovány (viz 3.1.5), byla přidána funkce programu, která dokáže vytvořit v kořenovém adresáři odkaz na jakýkoli adresář v souborovém systému. Je nutné pouze v proměnné `LINKED_DIRS` ve třídě Constants definovat název adresáře a cestu k odkazovanému adresáři.

3.4 Konfigurační soubory

Pomocí konfiguračních souborů lze nastavovat některé důležité proměnné.

- `groups.conf` – v tomto souboru je seznam všech mozných skupin. Každý název skupiny je na samostatném řádku příklad:

  ```
  dev-python
  sys-kernel
  app-office
  ...
  ```

- `main.conf` – soubor, kde se nastavuje připojení k databázi, úrověň logování atd., příklad:

  ```
  server=localhost
  dbname=portage
  user=root
  passwd=heslo
  loglevel=DEBUG
  ```

  - `server` – jméno nebo adresa databázového serveru
  - `dbname` – název databáze
  - `user` – jméno uživatele pro připojení k databázi
  - `passwd` – heslo uživatele pro připojení k databázi
  - `loglevel` – úroveň logování do souboru log.txt; možné hodnoty jsou DEBUG, INFO, WARN, ERROR, FATAL

- `groupfiles.conf` – v tomto souboru jsou definovány řetězce, kterými může začínat obyčejný adresář v adresáři skupiny (viz 3.3.6), každý takový řetězec je na jednom řádku

3.5 Přeložení programu

K přeložení programu pomocí kompilátoru `g++` je použit soubor `Makefile` a utilita `make`. Pro úspěšné sestavení jsou nutné kromě standardní knihovny C++ následující knihovny:

- log4cxx verze 0.9.7 – knihovna pro logování (viz 3.3.1)
- libpqxx verze 2.6.8– knihovan pro připojení k databázi
- fuse verze 2.6.4 – knihovna pro komunikaci s modulem FUSE v jádře

3.6 Spuštění programu

Program se spouští pouze s jedním parametrem, kterým je cesta k adresáři, který je přípojným bodem. Přiklad spuštění:

```
./dbp /usr/portage/layman/local/eclipse
```

Po spuštění se načtou konfigurační soubory a provede se inicializace. Program provádí operace s databází a pomocí namapovaných funkcí knihovny fuse je převádí na operase souborového systému. Jejich průběhy jsou zaznamenévány do logovacího souboru `log.txt`.

3.7 Specifikace testů

Ke správné funkci a spolehlivosti programu je nutné provést některé testy:

1. Vložení do souboru větší množství dat než je maximální povolená hodnota – operace bude odmítnuta a záznam o této události bude zapsán do logu
2. Zkopírování adresářové struktury a srovnání obsahu souborů a adresářů – veškeré adresáře a soubory včetně obsahu se musí zkopírovat správně aby při jejich porovnání utilitou `diff` nebyl zaznamenán rozdíl
3. Smazání adresářové struktury a zkontrolování obsahu databázových tabulek zdali došlo ke zrušení všech záznamů
4. Test na korektní funkci connection polu. (Opakované dlouhodobější mazání, kopírování souborů nebo přenos většího množství dat
5. Pokus o vytvoření souboru i když soubor se stejným jménem již existuje – operace bude odmítnuta a záznam o této události bude zapsán do logu
6. Pokus o vytvoření adresáře i když adresář se stejným jménem již existuje – operace bude odmítnuta a záznam o této události bude zapsán do logu
7. Spuštění při neexistujících konfiguračních souborech – nedojde ke spuštění programu a záznam o této události bude zapsán do logu
8. Spuštění s žádným nebo špatným parametrem – o ošetření této chyby se postará knihovna fuse, která vypíše na standardní chybový výstup varování

testovací počítač:	notebook HP nx7400
disk: procesor: jádro: referenční souborový systém:	Seagate 5400 ot./m Core 2 Duo 1,66GHz 2.6.21-gentoo-sources 64bit ReiserFS

Tabulka 3.1: testovací počítač

3.7.1 Měření výkonu

Testování probíhalo tak, že se do prázdného adresáře s připojeným FUSE souborovým systémem kopírovaly soubory a měřil se čas pomocí systémové utility time (viz man time). Stejným způsobem se měřilo i mazaní souborů. Veškeré naměřené hodnoty byly porovnány s hodnotami získanými na klasickém souborovém systému.

testovací data	celkový počet souborů	celkový počet adresářů	počet ebuildů	skupin	programů	typ operací	celkový čas	čas referenčního souborovém systému
overlay eclipse	73	23	19	1	11	cp	3,053	0,835
						rm	1,430	0,271
						sync	12,321	7.285
overlay python-head	164	90	6	2	6	cp	4,160	0,979
						rm	4,148	0,283
						sync	19,302	9,270

Tabulka 3.2: výkonostní testy; cp - kopírování dat, rm - mazání dat, sync - synchronizace

Nakonec se do portage přidal overlay. Obsah adresáře byl vymazán a připojil se na něj testovaný, souborový systém založený na FUSE. Poté byli pomocí systémové utility time měřeny jednotlivé doby trvání operací nad tímto adresářem.

Před každým měřením byla příkazem echo `3 > /proc/sys/vm/drop_caches` vyprázdněna vyrovnývací paměť souborového systému. Testovacími daty byly dva menší overlaye `eclipse` a `pythonhead`. Protože každý dílčí test byl proveden pouze třikrát a z těchto hodnot se spočítal průměr, výsledky testů jsou pouze orientační. A vzhledem k absenci vyrovnávací paměti by důkladnější měření neměla větší smysl. Parametry testovací sestavy jsou uvedeny v tabulce 3.1.

Kapitola 4

Závěr

Výsledkem této bakalářské práce je program, který dokáže uchovávat data stromu Portage v relační databázi a zpětně je zpřístupnit jako klasický souborový systém.

Byly provedeny všechny navržené testy a program byl odladěn tak, aby proběhly správně a podle předpokladů. Dále bylo provedeno orientační výkonostní srovnání s klasickým souborovým systémem. Souborový systém založený na relační databázi a FUSE je celkově pomalejší (viz tabulka 3.2). Vzhledem k tomu, že není implementována žádná cache, nejsou výsledky překvapující. I když při testech byla vyprázdněna vyrovávací paměť referenčního souborového systému, tak situace není srovnatelná, protože pokud si souborový systém nějaký soubor znovu načte, tak jej má v cache již k dispozici až do konce testu. Také není použita žádná embedded databáse, ale plnohodnotný databázový server.

Bylo by vhodné ebuildy také parsovat a tyto informace ukládat zvlášť do databáze. Zvýšení rychlosti by také mohlo přínést použití embedded databáze například SQLite[17][7] nebo Firebird[1]. Až několikanásobné zrychlení by mohla přinést implementace cache protože některé databázové dotazy jsou provéděny několikrát za sebou aniž by se data v databázi měnila. V ideálním případě by se každý vytvořený objekt mohl ukládat do vyrovnávací paměti, kterou může být asociativní pole map. Také by se měla použít ještě jedna vyrovnávací paměť pro metodu getattribs (viz 3.1.7), kam by se zase ukládaly informace o načtených souborech. V obou případech by klíčem byla cesta k souboru. A také některé metody objektů by bylo možno lépe optimalizovat.

Seznam použitých zkratek

- FUSE – Filesystem in Userspace [8]
- GNU – projekt zaměřený na svobodný software, rekurzivní zkratka pro GNU's Not Unix (GNU není Unix)
- gcc – The GNU Compiler Collection, GNU sada kompilátorů
- GPL – General Public Licence, Obecná veřejná licence
- ACID – Atomicity, Consistency, Isolation, Durability
- MIT – Massachusetts Institute of Technology
- LGPL – Lesser General Public License
- JDBC – Java Database Connectivit
- ODBC – Open Database Connectivity

Literatura

[1] Carlos H. Cantu. Get to know firebird in 2 minutes. http://www.firebirdnews.org/docs/fb2min.html.

[2] ENCYKLOPEDIE. Wikipedie - otevřená encyklopedie (gentoo linux). http://cs.wikipedia.org/wiki/GentooLinux.

[3] ENCYKLOPEDIE. Wikipedie - otevřená encyklopedie (linux). http://cs.wikipedia.org/wiki/Linux.

[4] ENCYKLOPEDIE. Wikipedie - otevřená encyklopedie (postgresql). http://cs.wikipedia.org/wiki/PostgreSQL.

[5] ENCYKLOPEDIE. Wikipedie - otevřená encyklopedie (relační databáze). http://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze.

[6] ENCYKLOPEDIE. Wikipedie - otevřená encyklopedie (sql). http://cs.wikipedia.org/wiki/SQL.

[7] Martin Lebeda. Sqlite - ultra lehké sql. http://www.root.cz/clanky/sqlite-ultra-lehke-sql/.

[8] WWW stránky. Fuse: Filesystem in userspace. http://fuse.sourceforge.net/.

[9] WWW stránky. Gentoo development guide. http://devmanual.gentoo.org/.

[10] WWW stránky. Gentoo linux use variable descriptions. http://www.gentoo.org/dyn/use-index.xml.

[11] WWW stránky. Gentoo upgrading guide. http://www.gentoo.org/doc/en/gentoo-upgrading.xml.

[12] WWW stránky. The gnu hurd - gnu project - free software foundation (fsf). http://www.gnu.org/software/hurd/hurd.html.

[13] WWW stránky. log4cxx. http://logging.apache.org/log4cxx/index.html.

[14] WWW stránky. Paludis, the other package mangler. http://paludis.pioto.org/overview/index.html.

[15] WWW stránky. pgfoundry. http://pgfoundry.org/.

[16] WWW stránky. Postgresql. http://www.postgresql.org/.

[17] WWW stránky. Sqlite. http://www.sqlite.org/.

Seznam příloh

- Příloha A – Obsah přiloženého CD
- Příloha B – SQL skript pro vytvoření databáze
- Příloha C – Metriky kódu

Přílohy

A Obsah přiloženého CD

- kořenový adresář – obsahuje soubor `readme.txt` a `Makefile` + konfigurační soubory
- adresář `src` – soubory se zdrojovými kódy a Makefile pro sestavení programu
- adresář `doc` – programová dokumentace v HTML formátu vygenerovaná pomocí programu doxygen
- adresář `sql` – obsahuje soubor `create_database.sql` se skriptem pro vytvoření databázových tabulet v databázi PostgreSQL
- adresář `BC` – bakalářská práce ve formátu pdf

B SQL skript pro vytvoření databáze

```
create table programGroup (
id serial primary key,
name varchar(30) not null,
ctime bigint,
mtime bigint
);
create unique index programGroup_idx on programGroup(id);

create table program (
id serial primary key,
groupId integer references programGroup(id),
name varchar(30) not null,
ctime bigint,
mtime bigint
);
create unique index program_idx on program(id);

create table ebuild (
id serial primary key,
programId integer references program(id),
version varchar(20) not null,
body varchar(500000),
ctime bigint,
mtime bigint
);
create unique index ebuild_idx on ebuild(id);

create table files (
id serial primary key,
parentDir integer references files(id),
pTypeId integer,
path varchar(256) not null,
name varchar(256) not null,
ftype varchar(20),
ptype varchar(20),
body varchar(500000),
ctime bigint,
mtime bigint
);
create unique index files_idx on files(id);

create table eclass (
id serial primary key,
name varchar(30),
body varchar(500000),
ctime bigint,
```

```
mtime bigint
);
create unique index eclass_idx on eclass(id);

--kořenový adresář portage
insert into files values (DEFAULT, NULL, NULL, '/', '/',
'DIR', 'ROOT', NULL);

--kořenový adresář eclass
insert into files values (DEFAULT, NULL, NULL, '/eclass/',
'/','DIR','ECLASS',  NULL);
```

C Metriky kódu

Velikost spustitelného souboru: 336,1KB
(systém Gentoo Linux 64 bitová architektura, při překladu bez ladicích informací)

Počet souborů : 18

soubor	řádků
Core.h	364
Constants.h	140
Exception.h	74
Portage.h	1266
util.h	49
Constants.cpp	172
Core.cpp	734
Ebuild.cpp	226
Eclass.cpp	242
Exception.cpp	45
FileBasedPortageElement.cpp	78
File.cpp	473
Group.cpp	345
Portage.cpp	925
PortageElement.cpp	195
Program.cpp	393
util.cpp	109

Tabulka 4.1: Metriky kódu