VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

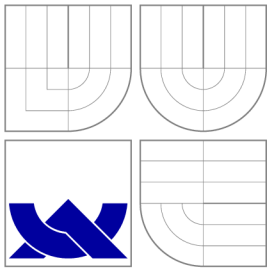
KOMPILOVACÍ SLUŽBA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

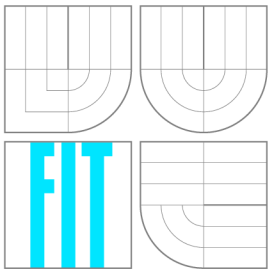
AUTOR PRÁCE
AUTHOR

LUKÁŠ TÍNES

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KOMPILOVACÍ SLUŽBA

COMPILE SERVICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ TÍNES

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. DAVID MARTINEK

BRNO 2009

Zadání bakalářské práce/9013/2008/xtines00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2008/2009

Zadání bakalářské práce

Řešitel: **Tínes Lukáš**
Obor: Informační technologie
Téma: **Kompilovací služba**
Kategorie: Softwarové inženýrství

Pokyny:

1. Prostudujte problematiku systémů pro vzdálený přístup k výpočetním zdrojům počítače bez nutnosti vlastnit na něm účet.
2. Navrhněte systém s webovým rozhraním, který uživatelům umožní využívat kompilátor GCC bez nutnosti vlastnit účet s přístupem k shellu. Systém musí umožňovat plánování činností a týmovou spolupráci na jednotlivých projektech. Proveďte analýzu bezpečnostních rizik takového systému a získané poznatky uplatněte při vlastním návrhu aplikace.
3. Implementujte navržený systém a otestujte jeho možnosti.
4. Zhodnoťte dosažené výsledky a navrhněte další možné směry vývoje.

Literatura:

- Gamma E.: Design patterns : elements of reusable objects-oriented software, Addison-Wesley, Boston, 1995.
- Podle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

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

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

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

Vedoucí: **Martinek David, Ing.**, UITS FIT VUT
Datum zadání: 1. listopadu 2008
Datum odevzdání: 20. května 2009

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

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tento dokument popisuje návrh a implementaci linuxové služby, která umožňuje přes webové rozhraní vzdálený překlad zdrojových kódů. Aplikace za použití překladového systému GNU make v jednotném a kontrolovaném prostředí přeloží tento kód do binární podoby. Obsahuje též podporu pro týmovou práci, plánování a má široké možnosti konfigurace. Celkově dovoluje uživateli překlad bez nutnosti vlastnit shell účet, případně mít jakýkoliv přímý přístup k serveru.

Abstract

This thesis describes design and implementation of Linux daemon which enables remote compiling of source files via web interface. Application creates binary files from sources using GNU make as a build platform in common and controlled environment. Implementation includes team support, planning and flexible configuration options. In the end, it enables users to compile without the need of having a shell access to server or any kind of direct access.

Klíčová slova

linux, služba, překlad, vzdálená kompilace, web, gcc, make

Keywords

linux, service, daemon, remote compilation, web, gcc, make

Citace

Lukáš Tínes: Kompilovací služba, bakalářská práce, Brno, FIT VUT v Brně, 2009

Kompilovací služba

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Davida Martinka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Tínes
19. května 2009

Poděkování

Chtěl bych poděkovat svému vedoucímu bakalářské práce Davidu Martinkovi za jeho odbornou pomoc

Obsah

Kapitola 1

Úvod

Tento dokument popisuje návrh a implementáciu systému s webovým rozhraním, ktoré umožní užívateľom jednoduchý prístup k službám vzdialeného prekladu zdrojového kódu pomocou prekladača GCC.

Pri práci na tímových projektoch je dôležité, aby všetci zainteresovaní členovia mali prístup k referenčným prekladom aplikácie. V prípade, že si každý vývojár kompiluje vlastnú verziu, môžu nastať chyby spôsobené inými verziami nástrojov, prípadne iným prostredím. Tak isto aj testeri môžu nahlasovať chyby spôsobené prekladom samotným. Tieto zdržania môžu značne skomplikovať projekt. A práve toto sa snaží odstrániť tento projekt.

Tento systém umožňuje jednotnú správu kompilácií a vďaka kontrolovanému prostrediu (verzie kompilačných nástrojov, knižníc, ...) zaručuje, že testeri sa môžu spoľahnúť, že chyby v aplikáciach neboli spôsobené kompiláciou samotnou. Taktiež dáva všetkým členom tímu prístup k výsledkom s rôznymi nastaveniami kompilácie.

Dôraz je kladený na tímovú spoluprácu, bezpečnosť a pohodlné užívateľské prostredie. Aplikácia je koncipovaná ako linuxová služba (tzv. *daemon*), ktorá bude bežať na pozadí a bude sa ovládať cez web rozhranie.

Využitie tejto aplikácie vidím najmä v malých a stredných firmách, kde vynikne možnosť tímovej spolupráce a centrálna administrácia. Taktiež je možné použitie v akademickej sfére, kde systém ponúka referenčnú platformu pre kompilovanie projektov.

V nasledujúcej kapitole bude oboznámenie s problematikou vzdialeného prístupu k zdrojom, s upriamením sa na vzdialený preklad zdrojového kódu. Ďalej budú spomenuté existujúce projekty s podobným zameraním. V tretej kapitole, pomenovanej Návrh aplikácie je popísaná celková architektúra systému. V kapitole Implementácia je popísané výsledné riešenie.

Kapitola 2

Systémy pre vzdialený prístup k zdrojom

2.1 Úvod

Potreba systémov pre vzdialený prístup k zdrojom vyplýva z náročnosti výpočtov a bezpečnostných obmedzení, ktoré bránia v priamom prístupe k týmto zdrojom (napr. vlastníctvo účtu). Tieto systémy rôznymi prostriedkami (či už verejné API[1], alebo cez užívateľské rozhranie) poskytujú svoj výpočtový výkon. Keďže táto práca je zameraná na vzdialený preklad, budem sa zaoberať najmä takýmito systémami. Systémy na vzdialený preklad sú väčšinou distribuované systémy s hierarchickou štruktúrou. V tejto štruktúre existuje jeden centrálny element (v niektorých prípadoch to môže byť priamo klient, napr. distcc), ktorý následne rozdeľuje úlohy medzi dostupné kompilovacie servery. Hlavným cieľom týchto systémov je najmä urýchlenie kompilácie veľkých projektov.

Tieto systémy umožňujú presunúť celú alebo určité časti kompilácie na vzdialený počítač a následné poskytnutie výsledku. Dôležitou vlastnosťou sú tiež informácie o priebehu kompilácie.

2.2 Distribúcia výpočtov

Distribuované systémy sú systémy s viac ako jedným výpočtovým alebo úložným elementom, na ktorom sa vykonávajú paralelné výpočty vo viac alebo menej kontrolovanom režime [7]. Pri týchto systémoch je dôležité rozdeliť úlohu na čo najmenšie logické časti, ktoré je potom možné paralelne vykonávať, väčšinou jednotlivé objektové súbory. Tým vzniká možnosť nezávislej paralelnej kompilácie jednotlivých súborov a následne ich spoločné linkovanie.

2.3 Bezpečnosť

Najväčšie bezpečnostné riziko predstavujú spúšťané úlohy. V rámci úlohy môže útočník namiesto kompilácie programu spustiť praktický akýkoľvek kód. Preto je nutné zaručiť izoláciu vykonávanej úlohy od kompilačného servera. Toto zaručí, že útočník bude mať prístup iba dátam, ktoré sú mu poskytnuté v rámci kompilačného prostredia.

[1] Application programming interface

Zároveň ale táto izolácia musí poskytovať možnosť komunikácie medzi procesom úlohy a procesom serveru, aby bolo možné túto úlohu ovládať a zaznamenávať jej priebeh.

Nasledujúce podkapitoly rozoberajú rôzne prístupy k tejto izolácii.

2.3.1 Chroot

Chroot je *POSIX* systémové volanie, ktoré zmení koreňový adresár procesu na iný ako je systémový. Zvyčajne je tento adresár rovnaký ako systémový – „/“. Toto systémove volanie práve tento adresár zmení na požadovaný. Na efektívne fungovanie tejto techniky je nutné pred samotným „uväznením“ zmeniť pracovný adresár procesu na adresár, v ktorom bude proces uväznený [6, 3].

Celý zmysel tohoto volania je transparentne uzamknúť proces v jednom adresári, takže prípadný útočník sa nedostane k súborom mimo neho.

Jedna z podstatných zraniteľností tejto metódy spočíva v prípadných otvorených súboroch. Ak má proces otvorený súbor mimo adresára, v ktorom je uväznený, je možné sa z tohoto väzenia dostať. Toto neplatí pre BSD systémy, ktoré majú inú, prepracovanejšiu implementáciu tohoto volania [3].

Ďalšou zraniteľnosťou je spôsob implementácie vo väčšine Unixových systémoch. V tomto prípade sa v informáciach o procese ukladá iba posledný koreňový adresár. Takže v prípade, že sa v uväznenom adresári znova zavolá *chroot*, tak sa koreňový adresár prepíše.

Kombináciou týchto dvoch zraniteľností sa je možné dostať z takéhoto uväznenia, ako je to ukázané v [3]. Základnou myšlienkou takéhoto útoku je postupná zmena pracovného adresáru na adresár o úroveň vyšší od aktuálneho (využíva sa špeciálny adresár „..“, ktorý sa odkazuje na nadradený adresár), až kým sa nedosiahne reálny systémový koreňový adresár. Účinnou ochranou pred takýmto útokom, je úprava súborového systému, kedy adresár „..“ odkazuje na adresár „.“, ktorý predstavuje odkaz na aktuálny adresár [5]. Toto ale predstavuje zásah do konzistencie súborového systému.

Spoločným znakom skoro všetkých zraniteľností tejto metódy je nutnosť mať práva super užívateľa (UID 0) [6]. V prípade, že uväznený proces má práva obyčajného užívateľa, sú možnosti prelomenia takého uväznenia značne obmedzené.

Na získanie práv super užívateľa sa najčastejšie používa exploit[2] na eleváciu práv procesu, prípade využitie súboru s nastaveným setuid bitom[3].

Komunikácia medzi úlohou a serverom

Pri použití tejto metódy, je možné použiť nástroje medziprocesovej komunikácie. Unixové systémy ponúkajú široké možnosti takejto komunikácie ako je zdieľaná pamäť, rúry alebo sockety [11].

2.3.2 Virtualizácia

Ďalším riešením zabezpečenia je kompletné virtualizovanie prostredia, v ktorom beží výpočet. V prípade virtualizovania, samotná kompilácia beží v úplne oddelenom prostredí, ide teda o oddelený operačný systém, ktorý beží na virtualizovanom hardvéri.

V prostredí Linuxu existuje niekoľko implementácii takéhoto virtualizovania. Medzi najznámejšie open-source patrí *Xen*, *kvm* a *QEMU*. Najpoužívanejší je ale *Xen*.

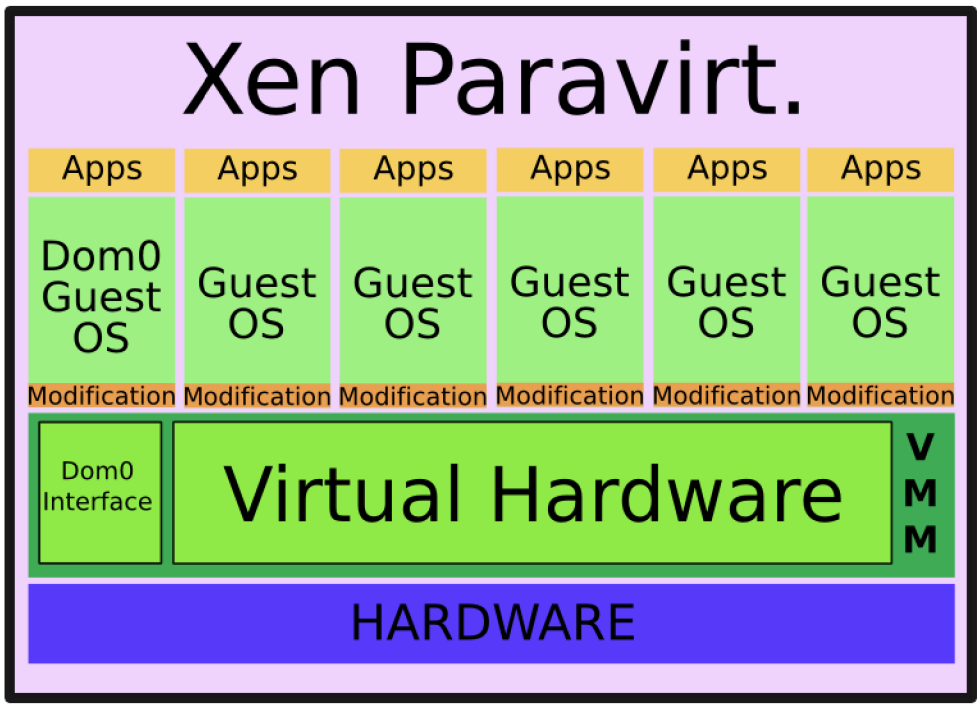
[2] malý program, ktorý zneužíva konkrétnu chybu OS

[3] špeciálny bit, ktorý pri spustení takéhoto súboru nastaví efektívneho užívateľa procesu na vlastníka takéhoto súboru (zvyčajne *root*)

V prípade *Xenu* ide vlastne o monitor virtuálnych strojov (ďalej len VS), ktorý virtualizuje hardvér a zabezpečuje izoláciu zdrojov jednotlivých VS. Z architektúry na Obrázku 2.1 vyplýva nutnosť podpory zo strany jadra virtualizovaného, ako aj hostiteľského operačného systému. Celá implementácia ma hierarchickú štruktúru, kde existuje jeden centrálny VS, nazývaný *Domain0*, ktorý ma väčšie práva a riadi ostatné VS. Tiež spravuje reálny hardvér [4]. Na vytvorenie VS je tiež nutné mať obraz disku takéhoto systému s nainštalovaným systémom, ktorý bude bežať vo VS.

Tento systém ponúka jednoduchú možnosť vytvárať a spravovať VS na jednom reálnom stroji.

Medzi hlavné nevýhody patrí zložitejšia konfigurácia a nutnosť značnej podpory na hostiteľskom stroji, keďže aj ten beží ako VS. Ďalšou nevýhodou je zložitá komunikácia medzi kompilačným serverom a samotnou úlohou, pretože úloha nielenže beží v inom procese, ale beží na inom stroji. V tomto prípade sa nedajú použiť žiadne nástroje priamej medziprocesovej komunikácie.



Obrázok 2.1: Architektúra Xenu

2.4 Jednotné prostredie

Pri tímovej práci často nastáva problém s rôznymi prostrediami (rôzne verzie knižníc, rôzne verzie prekladača, ...) jednotlivých členov a tým možnosť rôznych výsledkov prekladu (a tým aj neefektívneho testovania, keďže preklad rovnakého kódu sa môže odlišovať).

Vzdialený preklad tento problém rieši jednotným prostredím prekladu, ktorý sa pre každý preklad znovu vytvorí zo vzoru. Vďaka tomuto nie je prostredie prekladu ovplyvnené predošlým prekladom, čo jednak zaručuje vždy rovnaký preklad, a zároveň znižuje bezpečnostné riziko.

2.5 Podobné projekty

V nasledujúcej časti bude v stručnosti popísaných niekoľko projektov na vzdialený prístup k zdrojom. Vzhľadom na zamerenie tejto práce, výber bude obmedzený na projekty, ktoré ktoré sa zameriavajú, alebo aspoň obsahujú podporu pre vzdialené kompilovanie. Pri každom projekte bude spomenuté jeho zameranie a porovnanie kladov a záporov.

2.5.1 openSUSE BuildService

openSUSE BuildService je open-source projekt sponzorovaný firmou Novell. Celý projekt je zameraný na vytváranie binárnych balíkov pre rôzne linuxové distribúcie (zo začiatku iba pre openSUSE, neskôr rozšírené aj na ďalšie distribúcie) [12]. V rámci vytvárania balíkov prebieha tiež kompilácia.

Na zaručenie bezpečnosti hostiteľského serveru, samotná kompilácia prebieha vo virtualizovanom prostredí a tým od neho úplne oddelená. Virtualizovaný systém sa vytvára pri každom preklade, takže je zaručené vždy rovnaké prostredie pri kompilácii. Systém taktiež ponúka distribuovanie výpočtov medzi viacej serverov, nástroje na správu užívateľov a tímov a zaznamenávanie samotného prekladu, pre odhalenie chýb pri preklade.

Výhody

- Administrácia – systém má centrálnu administráciu užívateľov, tímov a projektov.
- Prístup – projekt ponúka verejné API, čo umožňuje integrovanie do ďalších projektov.
- Orientované ako služba – klient nemusí nič inštalovať na lokálnom PC (iba v prípade webového klienta).

Nevýhody

- Spätosť s balíkmi – preklad je iba jeden z krokov. Je nutné byť oboznámený s tvorbou balíkov a špecifikami jednotlivých distribúcií.

2.5.2 Distcc

Distcc je projekt distribuovaného prekladača gcc. Jeho základom je upravený prekladač gcc, ktorý rozdeľuje úlohy medzi jednotlivé servery. Výsledok kompilácie by mal byť rovnaký, ako lokálny preklad [8]. Celý preklad riadi klientská aplikácia, ktorá vykonáva predspracovanie (*preprocessing*), linkovanie ako aj ďalšie fázy. Vzdialená je iba samotná kompilácia.

Distcc bol pôvodne vyvinutý pre potreby open-source projektu Samba. Jeho potreba bola dôsledkom veľkosti zdrojového kódu projektu a s tým súvisiaci čas potrebný na kompiláciu.

Výhody

- Distribuovaný systém – rozdelenie výpočtov na viacej serverov.
- Rýchlosť – skoro až lineárne zrýchlenie vzhľadom na počet serverov.

Nevýhody

- Časť prekladu vykonáva klientská aplikácia – môžu nastať problémy s verziami knižníc pri práci v tíme.
- Žiadna autorizácia alebo bezpečnosť – neexistuje podpora pre správu užívateľov, skupín a vynucovanie obmedzení. Využívanie zdrojov je anonymné.
- Neexistencia centrálnej administrácie – keďže systém nie je centralizovaný, nie je možné centrálne spravovať prístup k serveru.
- Neexistencia GUI – neexistuje žiadne grafické rozhranie na zobrazenie priebehu jednotlivých úloh.

2.5.3 Cabie

Cabie (*Continuous Automated Build and Integration Environment*) je open-source systém na preklad napísaný Perle s web rozhraním. Má centrálnu administráciu úloh a užívateľov, podporu pre CM systémy ako CVS, Subversion a Perforce a notifikácie napríklad cez e-mail, a taktiež podporu pre testovanie [15].

Výhody

- Integrácia s CM – preklad aktuálnej verzie zdrojových súborov.
- Nightly builds – pravidelný automatický preklad najnovšej revízie projektu.
- Centrálna administrácia – možnosť sledovania priebehu prekladu cez webové rozhranie.
- Notifikácia – informovanie pomocou e-mailu alebo RSS o výsledku prekladu. Aplikácia obsahuje aj podporu pre mailing-listy.
- Podpora viacerých serverov – možnosť distribuovania výpočtu na viacej serverov s centrálnym úložiskom úloh a ich statusu.

Nevýhody

- Napísaný v Perle – použitím skriptovacieho jazyka riešenie stráca na výkone.
- Zložitá inštalácia – keďže neexistuje inštalátor, nasadenie tohoto riešenia môže byť značne obtiažne [9].
- Neexistencia tímov – existujú iba užívatelia.

2.6 Zhrnutie

Z popísaných projektov sa najviac môjmu zámeru približujú projekty *openSUSE BuildService* a *Cabie*. Svojimi zaujímavými časťami mi poslúžili ako inšpirácia.

openSUSE BuildService má prepracované užívateľské prostredie a správu užívateľov. Ponúka užívateľom prehľad o ich projektoch a úlohách. Touto vlastnosťou som sa inšpiroval pri tvorbe užívateľského rozhrania.

Cabie je zaujímavý svojou jednoduchou modulárnou architektúrou a možnosťami. Touto všestrannosťou a jednoduchosťou sa inšpiroval aj môj návrh architektúry.

Môj projekt si dáva za cieľ spojiť z týchto projektov tieto zaujímavé vlastnosti. To znamená jednoduchá a všestranná architektúra, otvorená prípadným rozšíreniam v budúcnosti a prepracovaná administrácia, či už pre správcov, vlastníkov projektov, alebo pre obyčajných užívateľov.

Kapitola 3

Návrh aplikácie

3.1 Základné ciele

Pred samotným návrhom celého projektu je nutné si presne určiť základné ciele, ktoré musí výsledná aplikácia mať. V nasledujúcom texte budú stručne opísané.

3.1.1 Kompilácia zdrojového kódu

Kompilácia je fundamentálna časť systému. Predpokladá sa podpora pre GNU Make ako systému pre kompilovanie projektov. S kompiláciou súvisí aj uloženie zdrojových a binárnych súborov, ktoré musí systém zabezpečiť. Tak isto aj protokol o preklade je podstatnou súčasťou výsledku, pretože je jediným zdrojom informácií v prípade chyby počas kompilácie.

3.1.2 Administrácia

Jednou z požiadaviek na systém bola podpora tímovej práce. Systém musí podporovať administráciu užívateľov (registrácia nových, mazanie, atď.), a aj administráciu projektov (vytváranie a rušenie a príslušnosť užívateľov do projektov). Logicky je možné rozdeliť užívateľov do 2 skupín:

- Administrátori – majú prístup k nastaveniam celého systému, pridávajú a odoberajú užívateľov.
- Užívatelia – môžu vytvárať nové projekty a pristupovať k projektom, ktorých sú členmi. Na základne tohoto členstva sa ďalej rozdeľujú na:
  - Bez účasti na projekte – môže iba vytvárať nové projekty
  - Člen projektu – môže pristupovať k projektom, ktorých je členom. Môže v nich vytvárať nové úlohy, prípadne upravovať existujúce. Nemôže ale takýto projekt upravovať alebo zmazať.
  - Vlastník projektu – môže všetko to, čo môže člen projektu a naviac môže tieto projekty upravovať, mazať a pridávať a odoberať členov projektu.

3.1.3 Webové rozhranie

Keďže webové rozhranie bude jediným spôsobom komunikácie so systémom, je nutné aby bolo prehľadné a zároveň funkčne postačujúce. Musí zohľadňovať postavenie užívateľov

a ich príslušnosť k projektom. Rovnako musí dať administrátorom nástroje na analýzu stavu systému. Je tiež nutné aby bolo optimalizované pre všetky majoritné internetové prehliadače.

3.2 Bezpečnosť

Ako už bolo rozoberané v podkapitole 2.3, spúšťaný kód predstavuje isté bezpečnostné riziko. V prípade tejto implementácie, najväčšie riziko pramení z podpory prekladového systému GNU Make, ktorý namiesto príkazov na kompiláciu pomocou GCC, môže obsahovať príkazy, ktoré môžu kompromitovať hostiteľský systém.

Ako riešenie tohto problému je v tejto implementácii použité systémové volanie *chroot*, ktoré efektívne obmedzí dopad takéhoto útoku iba na umelo vytvorené prostredie, ktoré neobsahuje žiadne citlivé informácie.

Na zaručenie efektivity je nutné aby

- neboli otvorené žiadne súbory mimo uväzneného adresára – keďže jediným otvoreným súborom serveru je databáza, je nutné aby pred spustením úlohy bola zatvorená.
- proces úlohy nemal práva super užívateľa – keďže volanie chroot je obmedzené iba na procesy s efektívnym UID 0, je nutné aby proces, ktorý bude vykonávať samotnú kompiláciu mal znížené práva.
- vytvorené prostredie neobsahovalo súbory s nastaveným setuid bitom – tieto súbory umožňujú zmenu efektívneho UID, čo umožňuje zmenu efektívneho UID procesu.

Splnením týchto podmienok výrazne obmedzíme možnosť prelomenia uväznenia.

Ani toto riešenie však nie je neprekonateľné. V prípade, že útočník využije chybu v jadre OS a získa práva super užívateľa, nič mu už nebráni vo využití známych techník na prelomenie tejto ochrany, ktoré sú popísané v podkapitole 2.3.1.

Vyššiu mieru ochrany by poskytovala virtualizácia, ktorá ale nebola použitá z dôvodu zložitého zapracovania do projektu a nutnosti prístupu k stroju s podporou virtualizácie. Predstavuje to ale možnosť zlepšenia v budúcnosti.

3.3 Architektúra

Celý systém možno rozdeliť na dva nezávislé celky: server a web rozhranie. Server je ďalej možné rozdeliť na viacej modulov. Každý modul obsluhuje jeden aspekt serveru. Tým sa docieli prehľadnosť implementácie a zjednodušenie opravy chýb. Toto rozdelenie názorne ukazuje Obrázok 3.1.

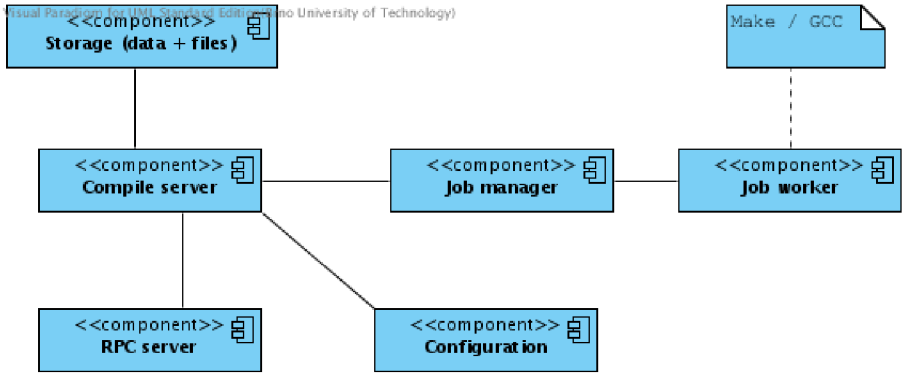
3.3.1 RPC server

Tento modul slúži na komunikáciu medzi samotným serverom a web rozhraním. Táto komunikácia prebieha pomocou RPC[1] volaní.

3.3.2 Compile server

Ústredný modul celej implementácie. Spája ostatné moduly do jednotného celku.

[1] Remote Procedure Call



Obrázok 3.1: Rozdelenie modulov

3.3.3 Job manager

Tento modul v pravidelných intervaloch kontroluje zoznam úloh a v prípade, že existuje čakajúca úloha, nájde zodpovedajúce zdrojové súbory a použije *Job worker*, ktorý vykoná samotnú úlohu.

3.3.4 Job worker

Modul, ktorý vykonáva samotnú úlohu. Ako už bolo spomenuté v podkapitole 3.2 oddelenie od *Job managera* je z bezpečnostných dôvodov. Tento modul bude bežať v procese so zníženými právami a izolovaný pomocou *chroot*. Po vykonaní úlohy sa ukončí.

3.3.5 Storage

Tento modul predstavuje perzistentné úložisko informácií o užívateľoch (prihlasovacie meno, heslo, práva), projektoch a úlohách. Rovnako obsahuje aj informácie o príslušnosti jednotlivých užívateľov k projektom a ich pozíciu v ňom.

3.3.6 Configuration

Tento modul udržiava informácie o aktuálnej konfigurácii servera. Tieto informácie sú platné iba pre konkrétnu inštanciu servera a po jeho ukončení sa stratia.

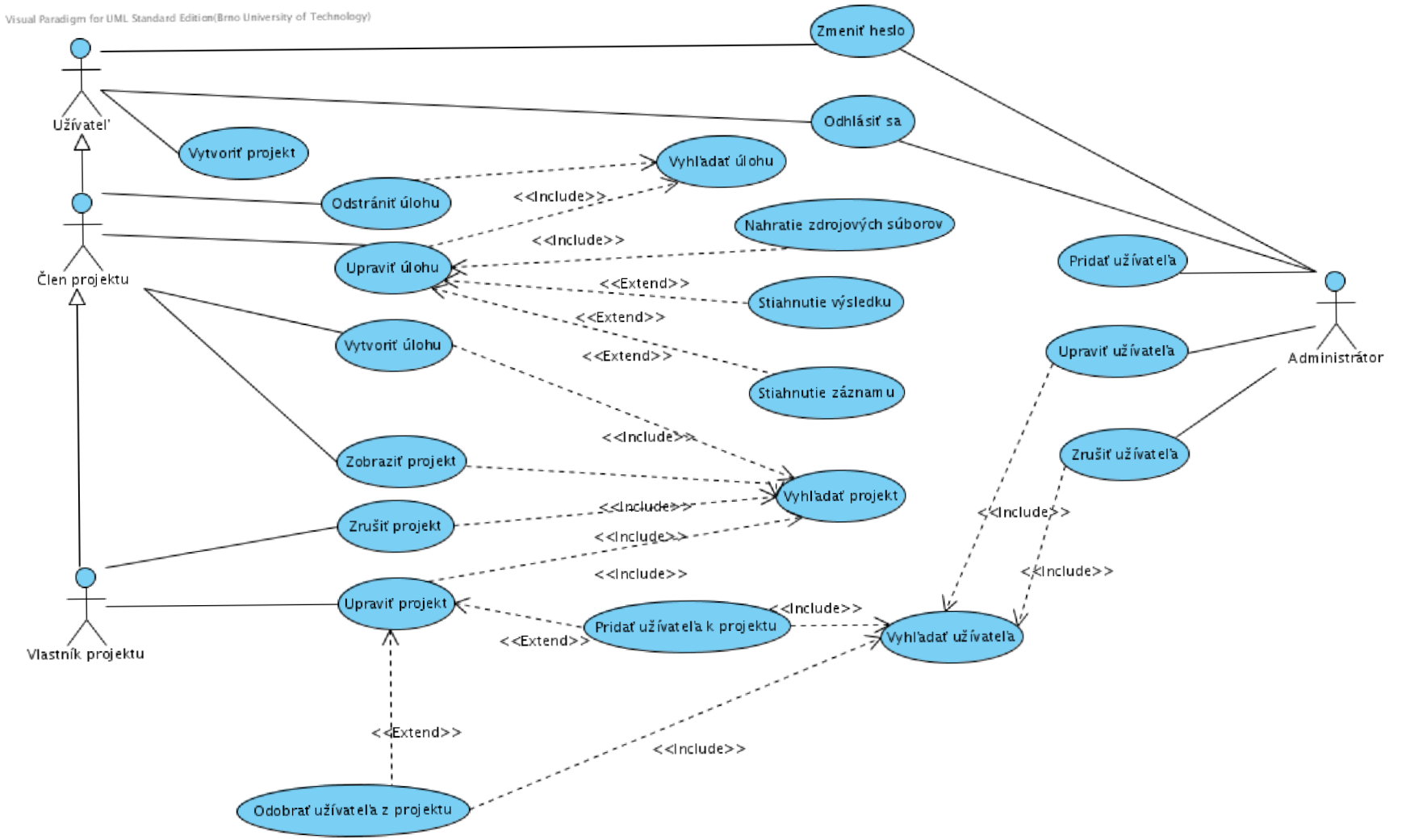
3.4 Use Case Diagram

Obrázok 3.2 predstavuje prípady užitia webového rozhrania. Predstavuje základ jeho návrhu a definíciu RPC protokolu. Z obrázku je zrejmé hierarchické usporiadanie užívateľov a ich obmedzenia popísané v podkapitole 3.1.2.

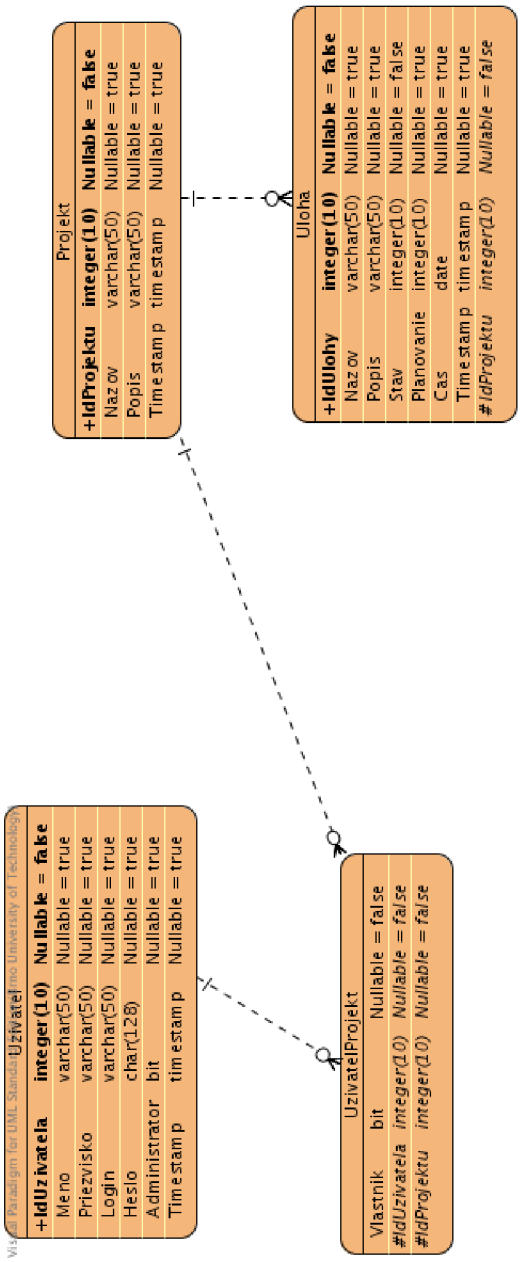
3.5 ER diagram

Obrázok 3.3 znázorňuje ER diagram[2] databázy, ktorá bude obsahovať perzistentné informácie o užívateľoch, projektoch a úlohách.

[2]Entity Relationship diagram



Obrázok 3.2: Use Case Diagram



Obrázok 3.3: ER diagram

Kapitola 4

Implementácia

Nasledujúca kapitola opisuje implementáciu systému podľa návrhu z predošlej kapitoly. Aplikácia je šírená pod licenciou EUPL[1], ktorá je kompatibilná s licenciou GPL. Táto licencia umožňuje bezplatné šírenie a úpravu zdrojových kódov, ale pritom chráni intelektuálne vlastníctvo. Plné znenie sa nachádza v súbore `EUPL Licencia.pdf` na CD.

Ako už z návrhu vyplýva, celá implementácia sa dá rozdeliť na štyri časti:

- Databáza
- RPC protokol
- Server
- Webové rozhranie

4.1 Použité technólogie

Na implementáciu serveru bol pre svoju rozšírenosť a široké možnosti vybraný jazyk C++. Využité sú aj pokročilé vlastnosti ako polymorfizmus a výnimky. Na implementáciu vlákien som použil knižnicu *pthreads*, ktorá je prirodzenou voľbou na POSIX systémoch.

Ako databázový stroj bol vybraný *SQLite* vďaka svojej jednoduchosti, rýchlosti a možnosti ho kompletne vstavať do aplikácie (odpadá nutnosť konfigurovať databázový server). Jednou z nevýhod je chýbajúca podpora pre vynucovanie obmedzení cudzích kľúčov, ktorá sa ale dá nahradiť triggermi [2].

Na komunikáciu medzi serverom a web rozhraním bol vybraný projekt *Thrift*, ktorý ma presvedčil svojou jednoduchosťou a možnosťou automaticky generovať obslužný kód [14].

Ako formát pre konfiguračný súbor bol vybraný jazyk XML[2], pretože je čitateľný pre ľudí a existuje mnoho knižníc slúžiacich na jeho spracovanie.

Na implementáciu web rozhrania bol použitý jazyk *PHP*. Na uľahčenie práce s JavaScriptom bola vybratá knižnica *jQuery*.

Na riadenie prekladu samotného projektu je použitý systém *CMake*, ktorý sa vyznačuje ľahkou konfiguráciou a podporou všetkých hlavných platforiem.

[1] European Union Public Licence
[2] Extensible Markup Language

4.1.1 Externé knižnice

Na implementovanie niektorých podporných funkcií boli použité externé knižnice. Všetky vybrané knižnice sú licencované ako open-source, aby nedochádzalo k licenčným problémom.

Použité sú tieto knižnice:

- SQLite – vstavaný databázový stroj, ktorý nepotrebuje konfiguráciu, ani samostatný server. Podporuje pokročilé vlastnosti ako transakcie, triggery a pohľady [1].
- Boost – je to knižnica, ktorá sa snaží byť implementáciu budúceho štandardu C++. Skladá z množstva knižníc od matematických funkcií až po vlákna a IPC[3] [10]. V tomto projekte sa využívajú tieto časti: `filesystem`, `lexical_cast` a `shared_ptr`.
- Crypto++ – táto knižnica predstavuje implementáciu veľkého množstva hashovacích a kryptografických funkcií. V tejto aplikácii sa využíva na hashovanie hesiel.
- TinyXML – táto odľahčená knižnica na prístup k XML súborom síce neposkytuje žiadne pokročilé funkcie, ale na prístup ku konfiguračnému súboru postačuje.
- Thrift – táto knižnica je určená na vytváranie medzijazykovej komunikácie. Umožňuje generovanie kódu na základe definície rozhrania [14].
- libtar – táto knižnica slúži na vytváranie .tar archívov. V tomto projekte sa používa na archiváciu.

4.2 Databáza

Návrh databázy priamo vychádza z ER diagramu na Obrázku 3.3. Väčšina tabuliek obsahuje položku TimeStamp, ktorá predchádza chybám pri race conditions[4].

4.2.1 Tabuľka uzivatel

Táto tabuľka predstavuje informácie o užívateľoch.

```
CREATE TABLE uzivatel(
             IdUzivatela INTEGER PRIMARY KEY NOT NULL,
             Meno VARCHAR(50),
             Priezvisko VARCHAR(50),
             Login VARCHAR(50),
             Heslo CHAR(128),
             Administrator INTEGER,
             TimeStamp DATE
)
```

4.2.2 Tabuľka projekt

Táto tabuľka predstavuje projekty a informácie o nich. Neobsahuje však priradenie užívateľov do projektov.

[3] Inter-process communication

[4] Náhodné chyby spôsobené paralelným prístupom k dátam.

```
CREATE TABLE projekt(
             idProjektu INTEGER PRIMARY KEY NOT NULL,
             Nazov VARCHAR(50),
             Popis VARCHAR(50),
             TimeStamp DATE
)
```

4.2.3 Tabuľka uloha

Tabuľka predstavujúca informácie o úlohách, ich stav a príslušnosť k projektu.

```
CREATE TABLE uloha(
             IdUlohy INTEGER PRIMARY KEY NOT NULL,
             Nazov VARCHAR(50),
             Popis VARCHAR(50),
             Stav INTEGER,
             IdProjektu INTEGER NOT NULL,
             Planovanie INTEGER DEFAULT 0,
             Cas INTEGER DEFAULT 0,
             TimeStamp DATE
)
```

Položka `Cas` v tomto prípade reprezentuje najskorší možný dátum vykonania úlohy, ale iba v prípade, že je `Planovanie` nastavené zodpovedajúcu hodnotu. Toto plánovanie môže byť: na najskorší možný čas, určený najskorší dátum.

4.2.4 Tabuľka uzivatel_projekt

Táto tabuľka popisuje priradenie užívateľov k projektom a ich pozíciu v ňom.

```
CREATE TABLE uzivatel_projekt(
             IdUzivatela INTEGER NOT NULL,
             IdProjektu INTEGER NOT NULL,
             Vlastnik INTEGER,
             )
```

4.2.5 Inštalácia

Pred prvým spustením servera je nutné vytvoriť čistú databázu. Tá sa vytvorí vykonaním

```
sqlite3 cs.db < db.sql
sqlite3 cs.db < fk.sql
```

kde `cs.db` je databázový súbor. Súbory `db.sql` a `fk.sql` sú SQL skripty dodané na CD. Tým sa vytvorí čistá databáza s jediným užívateľom *admin* (heslo *admin*).

4.3 RPC Protokol

Ako už bolo spomenuté vyššie, na komunikáciu medzi serverom a web rozhraním je použité RPC. Kontrola prístupu k jednotlivým funkciám sa vykonáva na strane rozhrania, to

znamená, že server nekontroluje tieto obmedzenia. Kompletný zoznam, popis a parametre funkcií, dátových štruktúr a výnimiek je v definičnom súbore `rpc.thrift` umiestnenom na CD. Z tohoto súboru sa dá následne vygenerovať kostra implementácie protokolu a to takto:

```
thrift -gen cpp -php rpc.thrft
```

4.4 Server

Serverová časť projektu je implementovaná ako linuxový daemon, ktorý beží na pozadí. Samotná aplikácia nemá žiadnu interakciu s užívateľom okrem konfiguračného súboru.

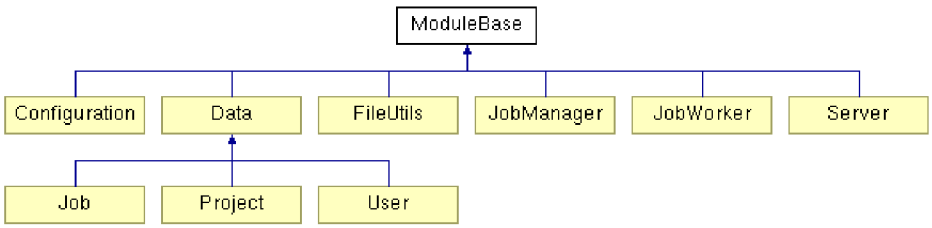
V súlade s návrhom sa server skladá z viacerých modulov. Väčšina z nich beží vo vlastnom vlákne.

4.4.1 Objektový návrh

Celá implementácia serveru je rozdelená na nasledujúce triedy:

- **Server** – centrálny modul serveru.
- **Configuration** – trieda predstavujúca aktuálnu konfiguráciu serveru.
- **JobManager** – trieda, ktorá hľadá dostupné úlohy a následne vytvorí *JobWorker*.
- **JobWorker** – trieda predstavujúca samotné vykonanie úlohy.
- **CServiceHandler** – trieda reprezentujúca RPC server, ktorý spracováva RPC volania.
- **Data** – trieda, ktorá je základom prístupu k databáze. Z tejto triedy ďalej dedia triedy na prístup k jednotlivým tabuľkám:
  - **Job** – trieda predstavujúca prístup k tabuľke *uloha*.
  - **Project** – trieda predstavujúca prístup k tabuľke *projekt*.
  - **User** – trieda predstavujúca prístup k tabuľke *uzivatel*.
- **FileUtils** – pomocná trieda obsahujúca funkcie na prístup k súborovému systému (vytváranie adresárov, archivácia a pod.).

Všetky tieto triedy dedia od triedy **ModuleBase**, ktorá obsahuje základné funkcie na ladenie. Znázornenie dedičnosti je na Obrázku 4.1



Obrázok 4.1: Diagram dedičnosti modulov

V ďalších častiach budú podrobnejšie opísané jednotlivé triedy. Vyčerpávajúci referenčný manuál k zdrojovým kódom je v súbore
`doc/Referenčný manuál.pdf` umiestnenom na CD.

4.4.2 Trieda ModuleBase

Táto abstraktná trieda predstavuje základ pre všetky triedy reprezentujúce moduly serveru. Poskytuje funkcie ako sú `DPrint` a `EPrint` na vypisovanie ladiacich informácii a chýb. Trieda tiež ponúka pomocnú funkciu na spracovanie regulárnych výrazov – `RegExec`.

Ukončovanie aplikácie

Keďže čisté ukončenie takejto aplikácie môže byť problém (prebiehajúcie úlohy, nedokončené RPC volania, ...), trieda poskytuje virtuálnu funkciu `SetQuit`, ktorá nastavuje príznak ukončovania. Preťažením tejto funkcie môžu zdedené triedy hierarchicky obsluhovať ukončenie aplikácie.

4.4.3 Trieda Server

Táto trieda je navrhnutá podľa *Singleton Design Pattern*. To znamená, že v celej aplikácii existuje iba jedna inštancia tejto triedy, ku ktorej ostatné triedy pristupujú. Zároveň nemá verejný konštruktor [13]. K inštancii sa pristupuje cez statickú funkciu `Instance`.

Táto trieda udržuje inštancie tried: **Configuration**, **JobManager** a **CServiceHandler**.

Implementuje tiež prepínanie aplikácie do pozadia pomocou funkcie `Deamonize` a spracovanie signálov funkciami `QuitSignalHandler` a `ReconfigureSignalHandler`. Spracovávané signály sú: SIGTERM, SIGINT a SIGHUP.

4.4.4 Trieda Configuration

Ako už bolo spomenuté, táto trieda obsahuje informácie o konfigurácii servera. Jednotlivé konfiguračné voľby sú tieto

- `RPCPort` – port na ktorom bude RPC server očakávať požiadavky. Východzia hodnota je 2222.
- `DbFile` – cesta k súboru, ktorý obsahuje SQLite databázu.
- `StorageRoot` – cesta k adresáru, do ktorého sa budú ukladať súbory úloh. Bude sa v ňom aj vytvárať dočasný adresár na kompiláciu.
- `ConfigFile` – cesta k XML konfiguračnému súboru.
- `Deamonize` – príznak, či sa má aplikácia po spustení prepnúť do pozadia. Východzia hodnota je `true`.
- `JobWait` – čas v sekundách udávajúci dobu čakania **JobManagera** na zopakovanie hľadania úlohy. Východzia hodnota je 5 sekúnd.
- `EnvScript` – cesta k skriptu, ktorý sa spúšťa pred každou kompiláciou a má za úlohu vytvorenie kompilačného prostredia. Bližšie sa o tomto skripte pojednáva v podkapitole 4.4.6.

- `ShowHelp` – príznak zobrazenia pomoci. Východzia hodnota je `false`.
- `User` – meno užívateľa, ktorý bude nastavený ako vlastník procesu, ktorý bude vykonávať samotnú kompiláciu.
- `UID` – získané id užívateľa podľa `User`.
- `Group` – meno skupiny, ktorá bude nastavená ako vlastník procesu, ktorý bude vykonávať samotnú kompiláciu.
- `GID` – získané id skupiny podľa `Group`.
- `Make` – cesta k make vzhľadom na kompilačný adresár. Interpretácia tejto cesty je závislá na nastavení `UseChroot`.
- `UseChroot` – určuje, či sa použije volanie *chroot* pred spustením úlohy. Toto opatrenie podstatne zvyšuje bezpečnosť serveru, ale je nutné aby proces serveru mal práva super užívateľa.

  Od nastavenia tejto voľby závisí aj interpretácia ciest k `make` a v samotnom Makefile úlohy. Vyplýva to zo samotnej podstaty volania chroot [6]. Táto voľba bola pridaná, aby bolo možné použitie aj bez spúšťania ako super užívateľ. Východzia hodnota je `true`.
- `HardQuit` – určuje spôsob ukončenia úlohy pri ukončení aplikácie. Ak je `false`, tak aplikácia počká na jej ukončenie. Východzia hodnota je `true`.

Trieda umožňuje spájanie dvoch konfigurácii pomocou funkcie `Merge`. Spájanie konfigurácii sa využíva na kombinovanie parametrov z príkazového riadku a z konfiguračného súboru.

Parametre z príkazového riadku

Niektoré základné nastavenia sa dajú nastaviť aj cez príkazový riadok pri spustení. Každý parameter má krátku (začínajúca sa s „`-`“), a aj dlhú (začínajúca sa s „`--`“) variantu. Spracovanie týchto parametrov prebieha vo funkcii `ReadCmdLineConfig`. Aplikácia rozoznáva tieto voľby:

- `-c` alebo `--config-file` – určuje cestu ku konfiguračnému súboru. Je to povinný parameter.
- `-p` alebo `--port` – určuje port RPC servera.
- `-n` alebo `--no-daemon` – zamedzí prechodu do pozadia pri spustení.
- `-h` alebo `--help` – zobrazí pomoc a ukončí sa.

Konfiguračný súbor

Konfiguračný súbor je XML súbor, ktorého popis nájdete v súbore `config.xml.default` na CD. Súbor je rozdelený na tri časti: server, jobmanager a rpc. Možné voľby sú skoro identické s konfiguračnými voľbami tejto triedy.

4.4.5 Trieda JobManager

Táto trieda spravuje jednotlivé úlohy. Jej hlavnou činnosťou je funkcia `Run`, prípadne `RunThread`, ktorá spustí nekonečný cyklus v ktorom pravidelne kontroluje úlohy a v prípade, že sa nájde úloha, ktorá spĺňa nasledujúce podmienky

1. ešte nebola spustená a nie je zakázaná.
2. je naplánovaná na čo najskoršie spustenie alebo je naplánovaná na určitý dátum a tento dátum už nastal.

tak sa vytvorí inštancia triedy **JobWorker** a úloha sa spustí. Kontrola ďalších úloh pokračuje až po ukončení úlohy.

4.4.6 Trieda JobWorker

Táto trieda predstavuje vykonávanie jednej úlohy, takže je spätá s jednou úlohou (premenná `_job`). Vykonanie úlohy prebieha v niekoľkých krokoch:

1. Vytvorenie prostredia na kompiláciu (`CreateEnvironment`) – tento krok zahŕňa vytvorenie adresárovej štruktúry dočasného adresára, spustenie skriptu na vytvorenie prostredia a skopírovanie zdrojových súborov. Skript je bližšie popísaný v podkapitole 4.4.6.
2. Vytvorenie procesu a spustenie úlohy (`DetachProcess`) – v tomto kroku sa vytvorí nový proces, uväzní sa a znížia sa jeho práva. Následne sa prepojí štandardný a chybový výstup s rodičovským procesom, aby bolo možné zaznamenávať proces kompilácie do záznamu. Technika tohoto prepojenia je popísaná ďalej v podkapitole 4.4.6. Následne sa spustí príkaz `make all`.
3. Pripravenie výsledku (`PrepareResult`) – tento krok predstavuje skopírovanie výsledných súborov do určeného adresára. Aj v tomto kroku sa vytvorí nový proces, uväzní sa, znížia sa mu práva a prepojí sa výstup. Následne sa spustí príkaz `make install`. Očakáva sa, že tento príkaz pripraví výsledky v adresári `install` (cesta je závislá na nastavení `UseChroot`).
4. Archivácia výsledku (`SaveBinary`) – ak sa kompilácia úspešne skončila, výsledok sa uloží do tar archívu.
5. Archivácia zdrojových súborov (`ArchiveSource`) – na konci sa pôvodné zdrojové súbory archivujú v tar súbore.
6. Upratanie prostredia (`CleanupEnvironment`) – vymazanie dočasného adresára.
7. Uloženie záznamu (`SaveLog`) – záznam sa zapíše do textového súboru.

Skript na vytvorenie prostredia

Tento skript je spúšťaný vždy pri vytváraní prostredia, teda pred každou kompiláciou. Tento skript dodávaný na CD, pretože závisí na konkrétnych potrebách administrátora.

Výsledkom by ale malo byť prostredie, ktoré bude obsahovať kompilačné nástroje, potrebné hlavičkové súbory a podobne. Ako jediný parameter sa posiela absolútna cesta k adresáru, kde sa má prostredie vytvoriť.

Vytváranie záznamu

Na hľadanie chýb pri kompilácii je užitočný záznam o kompilácii. Na zápis do záznamu sa používa makro `AddLogMessage`.

Na prepojenie výstupu vytvorených procesov s rodičovským je použitá *rúra*[5]. Ide o jednosmernú komunikáciu medzi procesmi, kde jeden proces zapisuje dáta do jedného konca rúry a ďalší proces z druhého konca číta tieto dáta [11].

V tomto prípade je prepojený štandardný a chybový výstup so zápisovým koncom rúry a rodičovský proces vo funkcii `AddLogMessageFD` postupne číta z druhého konca.

4.4.7 Trieda CServiceHandler

Táto trieda predstavuje implementáciu kostry, ktorá je generovaná z definície RPC protokolu.

V tejto aplikácii je konkrétne použitý `TNonblockingServer`, ktorý sa vyznačuje vysokým výkonom bez použitia vláken. Pridáva ale ďalšiu závislosť na externej knižnici – *libevent*.

RPC server sa spúšťa funkciou `DispatchServer`, ktorá vytvorí vlákno pre tento server, vytvorí ho a spustí.

Ukončovanie serveru

Keďže spravovanie RPC volaní je asynchrónne voči hlavnému vláknu, ktoré spracováva signály, je vytvorený spôsob ako čisto ukončiť všetky RPC volania pred ukončením serveru. Využíva sa na to funkcia `SetQuit` popísaná v 4.4.2.

Pred každým spravovaním RPC volania sa zavolá funkcia `Enter`, ktorá skontroluje, či sa server nevypína. Ak sa vypína, tak skontroluje počítadlo aktívnych volaní. Ak je 0, tak signalizuje čakajúcemu vláknu (vlákno, ktoré zavolalo `SetQuit`), že všetky volania boli ukončené. Nakoniec vráti výnimku. V prípade, že sa server nevypína, tak iba inkrementuje počítadlo aktívnych volaní – `Processing`. Na konci každého volania sa volá funkcia `Leave`, ktorá toto počítadlo dekrementuje.

Vlákno, ktoré zavolalo `SetQuit`, v prípade, že existuje aktívne RPC volanie (`Processing` $\neq 0$), čaká, kým sa neukončí posledné volanie. Tým je zaručené čisté ukončenie už začatých volaní a zamedzenie spracovávania nových.

4.4.8 Trieda Data

Táto trieda prestavuje základ prístupu k databáze. Spojenie s databázou sa uchováva v premennej `DbConn` a vytvára sa, respektíve ruší, pomocou funkcie `DbInit`, respektíve `DbDestroy`.

Základom tejto triedy je funkcia `ByQuery`, ktorá spracuje textový príkaz pre databázový stroj a cez virtuálnu funkciu `ByStatement` naplní hodnoty príslušnej odvodenej triedy. Vlastnosť `Filter` umožňuje transparentne pridávať ďalšie obmedzenia na vybraný súbor záznamov. Trieda má taktiež virtuálne funkcie `New`, `Save` a `Delete`, ktoré slúžia na vytváranie, ukladanie a mazanie záznamu.

V prípade, že `ByQuery` vráti viac ako jeden záznam, tak prvý záznam sa uloží do aktuálnej inštancie a ďalšie sa ukladajú ako nové inštancie vytvorené virtuálnou funkciou

[5] angl. pipe

`CreatePtr`. Tieto nové inštancie sa postupne ukladajú do poľa `Items`. Aby bol zjednotený prístup k jednotlivým položkám, je preťažený operátor `[]`, ktorý podľa indexu vracia príslušnú inštanciu (či už aktuálnu alebo z poľa `Items`).

Trieda obsahuje aj jednu pomocnú funkciu – `GetSHAHash`, ktorá vytvára SHA-512 hash. Táto funkcia sa využíva napríklad pri ukladaní hesla.

Ukladanie a konkurencia

Keďže aplikácia je viacvláknový, je nutné ošetriť konkurentný zápis dát. Na to sa využíva pole `TimeStamp` dátovej tabuľky, ktorý sa pred každým zápisom kontroluje s naposledy načítaným. Ak sa medzitým zmenil, znamená to, že záznam bol medzitým zmenený iným vláknom a nie je možné zmeny zapísať.

Konverzia na RPC štruktúry

Keďže štruktúry v RPC protokole sa líšia od tried použitých v aplikácii, každá trieda obsahuje 2 funkcie: `Fill` a `FillFrom`, ktoré vykonávajú konverziu medzi nimi.

4.4.9 Trieda Job

Táto trieda je odvodená od triedy **Data** a predstavuje jeden alebo viacej záznamov v tabuľke `uloha`. Naviac obsahuje aj funkcie na zistenie adresárov úlohy, ako: `GetSourcePath`, `GetCompilePath`, `GetBinaryPath`a `GetArchivePath`. Obsahuje tiež ďalšie funkcie na prístup k súborom ako `GetLog` alebo `Archive`.

Prístup k záznamom

Na prístup k záznamom slúžia tieto funkcie:

- `ByID` – naplní objekt podľa `IdUloha`.
- `ByProject` – naplní objekt podľa `IdProjektu`.
- `ByNotStarted` – naplní objekt všetkými nezačatými úlohami.

4.4.10 Trieda Project

Táto trieda predstavuje jeden alebo viacero záznamov v tabuľke `projekt`. Navyše ešte obsahuje aj funkcie na riadenie vzťahu medzi užívateľom a projektom:

Prístup k záznamom

Na prístup k záznamom slúžia tieto funkcie:

- `ByID` – naplní objekt podľa `IdProjektu`.
- `ByUser` – naplní objekt všetkými projektami, v ktorých je užívateľ aspoň členom.
- `ByJob` – naplní objekt projektom, ktorému patrí zadaná úloha.

4.4.11 Trieda User

Trieda predstavujúca jeden alebo viacero záznamov v tabuľke `uzivatel`.

Keďže heslo je v databáze hashované a nie je ho možné získať v pôvodnej podobe, po prečítaní sa napĺňa iba `HesloHash`. Ak však je pri zápise `Heslo` neprázdne, vytvorí sa z neho hash a zapíše sa toto nové heslo.

Prístup k záznamom

Na prístup k záznamom slúžia tieto funkcie:

- `ByID` – naplní objekt podľa `IdUzivatela`.
- `ByLoginPass` – naplní objekt podľa kontroly `Loginu` a `Hesla`.
- `ByProject` – naplní objekt všetkými užívateľmi, ktorí patria do zadaného projektu.
- `ByAll` – naplní objekt všetkými užívateľmi.

4.4.12 Trieda FileUtils

Táto trieda predstavuje zbierku statických funkcií, ktoré zaobaľujú prístup k súborovému systému pomocou knižnice `Boost::filesystem`.

4.4.13 Inštalovanie serveru

Inštalácia servera prebieha v dvoch krokoch.

Preklad aplikácie

Aplikácia sa prekladá štandardným kompilátorom gcc. Kompilácia bola testovaná na systémoch Linux. Pred samotným prekladom, je nutné nakonfigurovať tento preklad. To sa vykoná príkazom

```
cmake .
```

Jednotlivé voľby pre konfiguráciu prekladu sú popísané v súbore `INSTALL` na CD. Po nakonfigurovaní je možné spustiť preklad vykonaním

```
make all
```

A následne aplikáciu nainštalovať vykonaním

```
make install
```

V rámci inštalácie sa prekopíruje súbor `config.xml.default` do `/etc/cservice`.

Príprava pred spustením

Po tomto je ešte nutné upraviť konfiguračný súbor, ktorý je popísaný v 4.4.4. Najjednoduchšie je použiť vzorový súbor `config.xml.default` a ten si následne upraviť.

Inštalácia databázy je popísaná v 4.2.5 a inštalácie web rozhrania je popísaná v 4.5.3. Po nainštalovaní všetkých týchto častí je možné spustiť server vykonaním

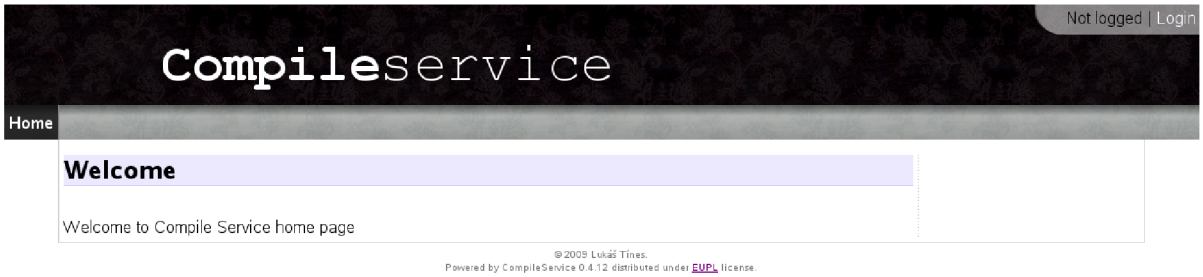
```
CompileService
```

Parametre príkazového riadku sú popísané v 4.4.4.

4.5 Web rozhranie

Návrh rozhrania vychádza z prípadov užitia znázornených na Obrázku 3.2. Dizajn tohoto rozhrania sa snaží byť čo najpríjemnejší ale zároveň prehľadný. Celé rozhranie je v angličtine, keďže sa u užívateľov predpokladá aspoň základná znalosť tohoto jazyka.

Rozhranie je napísané v jazyku PHP s využitím jazykov XHTML, CSS a JavaScript a je dodávané v adresári `php` na CD.

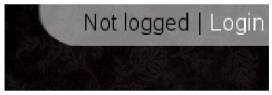


Obrázok 4.2: Uvítacia obrazovka web rozhrania

4.5.1 Rozhranie

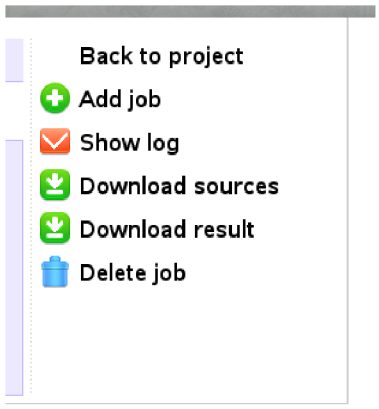
Ako vidno na Obrázku 4.2, rozhranie sa skladá zo 4 častí

- Stav prihlásenia – môže mať dva stavy: neprihlásený (Obrázok 4.3) a prihlásený (Obrázok 4.4).
- Hlavné menu – obsah tohoto menu sa mení podľa práv, ktoré ma prihlásený užívateľ.
- Obsah stránky – samotný obsah stránky. Mení sa podľa podstránky.
- Menu stránky – menu špecifické pre každú podstránku (Obrázok 4.5).



Obrázok 4.3: Stav prihlásenia – neprihlásený.

Obrázok 4.4: Stav prihlásenia – prihlásený.



Obrázok 4.5: Príklad menu stránky. Každá voľba je reprezentovaná aj graficky pre lepšiu orientáciu.

4.5.2 Architektúra

Aj implementácia rozhrania sa drží objektového princípu programovania a naplno využíva objektové vlastnosti PHP. Ako základná trieda je použitá abstraktná trieda **Page**, prípadne jej odľahčená verzia **PopupPage**, ktorá predstavuje základ pre stránky do vyskakovacích okien. Z týchto tried dedia všetky podstránky.

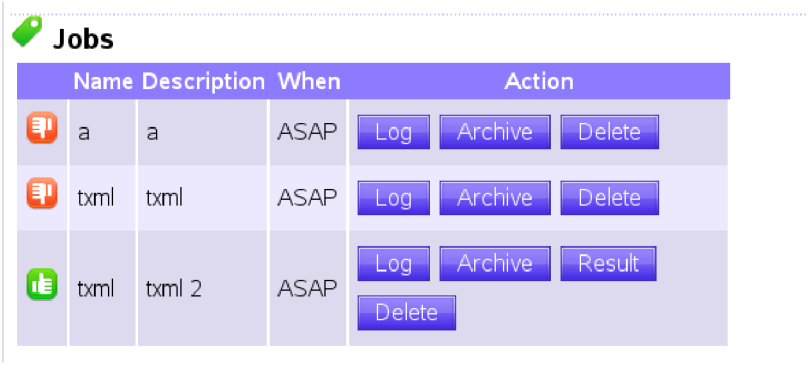
Väčšina objektov ako sú tabuľky, menu, tlačítka, formuláre a podobne sú zdedené z triedy **Object**.

Celá architektúra rozhrania je postavená na jednom súbore – `index.php`. V tomto súbore sa načítavajú parametre a kontroluje prístup k jednotlivým funkciám.

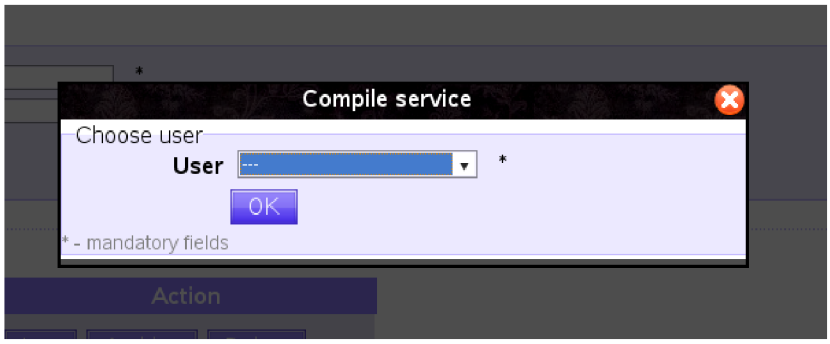
Podstatné sú však dva parametre: **mode** a **action**. Parameter **mode** udáva ktorá podstránka sa zobrazí a **action** určuje akciu, ktorá je špecifická pre každú podstránku. Jednotlivé podstránky sú tieto:

- **welcome** – uvítacia stránka. Taktiež stránka, ktorá zobrazuje chyby.
- **login** – prihlásenie sa do systému.
- **users** – zobrazí zoznam užívateľov.
- **user** – detail užívateľa, prípadne vytvorenie nového.
- **job** – detail úlohy z možnosťou nahratia zdrojových kódov. Umožňuje tiež vytvorenie novej úlohy.
- **projects** – zoznam projektov aktuálne prihláseného užívateľa
- **project** – detail projektu. Obsahuje tiež stav všetkých úloh (znázornené na Obrázku 4.6) a zoznam členov projektu.
- **stats** – štatistiky serveru.

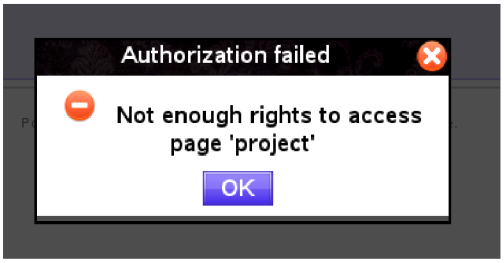
Ďalej sú tu podstránky, ktoré sa zobrazujú ako vyskakujúce okno: **choose_user** (Obrázok 4.7), **chpwd** a **message**(Obrázok 4.8). Špeciálnym prípadom podstránky je **job_redir**, ktorá iba presmeruje na stránku
`job_down.php`. Slúži to sťahovanie záznamu o kompilácii a archívov.



Obrázok 4.6: Príklad zoznamu úloh s ich stavmi. Každý stav je reprezentovaný špecifickou ikonou pre jednoduchú orientáciu.



Obrázok 4.7: Príklad výberu užívateľa do projektu.



Obrázok 4.8: Príklad vyskakovacieho okna s chybou.

4.5.3 Inštalácia

Na nainštalovanie tohoto rozhrania stačí prekopírovať adresár `php`, ktorý je umiestnený na CD, do adresára web serveru (napríklad *Apache*) a editovať súbor `thrift_conn.php`.

V tomto súbore je nutné nastaviť premennú `$thrift_host` na IP alebo DNS názov počítača, na ktorom beží RPC server. Výhodou je, že web rozhranie a server môžu bežať na dvoch oddelených počítačoch. Ďalej je treba nastaviť premennú `$thrift_port` na port, na ktorom RPC server očakáva požiadavky.

Kapitola 5

Testovanie

Testovanie riešenia bolo zamerané na overenie 2 základných vlastností: kompilácia zdrojového kódu a bezpečnosť. Testovanie prebiehalo so zapnutými bezpečnostnými prvkami ako *chroot* a zníženie práv. CD obsahuje v adresári `test` testovacie zdrojové kódy, ktoré boli použité na tieto testy.

5.1 Kompilácia zdrojového kódu

Na túto časť testovania bola použitá distribúcia projektu *TinyXML*. Očakávaným výsledkom tohoto testu bol výsledný spustiteľný súbor dostupný užívateľovi na stiahnutie. Tento výsledok sa podarilo dosiahnuť.

5.2 Bezpečnosť

Na testovanie bezpečnosti bol použitý exploit popísaný v [3]. Na overenie výsledku bol kód upravený, aby vytvoril súbor mimo pôvodného adresára. V rámci testovacieho *Makefile* bol tento kód skompilovaný a spustený. Očakávaným výsledkom tohoto testu bolo zlyhanie spustenia takéhoto kódu. Tento výsledok sa podarilo dosiahnuť.

Kapitola 6

Záver

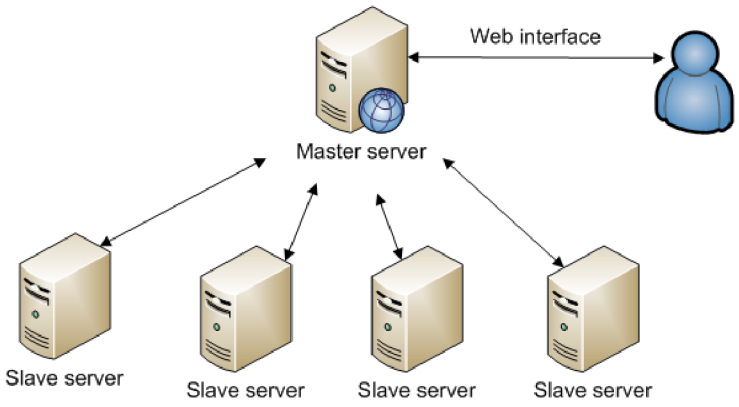
Celkovo môžem zhodnotiť výsledok tohoto projektu ako úspešný. Podarilo sa mi splniť všetky požiadavky vyplývajúce jednak zo zadania ako aj z návrhu aplikácie.

Serverová časť je prehľadná, rýchla a robustná, čo uľahčí jej budúce rozšírenie. Umožňuje kompilovať, plánovať toto kompilovanie a spravovať užívateľov a projekty.

Web rozhranie je jednoduché a vďaka využitiu grafických symbolov aj prehľadné. Spôsob implementácie tiež otvára cestu na rozšírenie a zlepšenie v budúcnosti.

6.1 Možné rozšírenia v budúcnosti

Najväčšiu možnosť rozšírenia vidím v umožnení distribúcie kompilovania na viacej podriadených serverov. Takáto architektúra je načrtnutá na Obrázku 6.1.



Obrázok 6.1: Distribúcia prekladu na viacej serverov.

Podstatným zlepšením bezpečnosti by bolo nahradenie *chroot* volanie virtualizáciou.

Ďalšie možné rozšírenie vidím vo vytvorení platformy na vytváranie zásuvných modulov, ktoré rozšíria možnosti o priamy import zdrojových kódov z (D)VCS[1], podporu iných prekladových systémov ako *make* a iné.

Web rozhranie je tiež možné rozšíriť na viacej jazykov.

[1] (Distributed) Version Control System – (distribuovaný) systém na správu verzií, ako git, svn, gh

6.2 Prínosy

Pri práci na tomto projekte som sa osvojil prácu s vláknami a programovanie RPC protokolu. Taktiež som si vyskúšal prácu so systémom na správu chýb, ktorý som počas vývoja používal. Dúfam, že táto práca pomôže zjednotiť a sprehľadniť kompilácie pri vývoji a testovaní softvéru.

Literatura

[1] *About SQLite*. [online], [rev. 2009-03-28], [cit. 2009-05-11].
URL http://www.sqlite.org/about.html

[2] *SQL Features That SQLite Does Not Implement*. [online], [rev. 2009-01-03], [cit. 2009-05-11].
URL http://www.sqlite.org/omitted.html

[3] Burr, S.: *How to break out of a chroot() jail*. [online], [rev. 2002-05-12], [cit. 2009-05-12].
URL http://www.bpfh.net/simes/computing/chroot-break.html

[4] Buytaert, K.: *Linux Virtualization with Xen*. [online], [rev. 2006-01-26], [cit. 2009-05-12].
URL http://www.linuxdevcenter.com/pub/a/linux/2006/01/26/xen.html

[5] Fennelly, C.: Wizard's Guide To Security: Summertime potluck. *SunWorld Online*, August 1999.

[6] Friedl, S. J.: *Best Practices for UNIX chroot() Operations*. [online], [rev. 2002-01-18], [cit. 2009-01-20].
URL http://www.yolinux.com/TUTORIALS/CabieBuildSystem.html

[7] Godfrey, B.: *A primer on distributed computing*. [online], [cit. 2009-01-27].
URL http://www.bacchae.co.uk/docs/dist.html

[8] Henderson, F.: *distcc: a fast, free distributed C/C++ compiler*. [online], [cit. 2009-01-06].
URL http://code.google.com/p/distcc/

[9] Ippolito, G.: *Linux Tutorial: Cabie Automated Build System*. [online], [cit. 2009-05-10].
URL http://www.yolinux.com/TUTORIALS/CabieBuildSystem.html

[10] Karlsson, B.: *Beyond the C++ Standard Library: An Introduction to Boost*. Addison Wesley Professional, 2005, ISBN 0321133544.

[11] Mitchell, M.; Oldham, J.; Samuel, A.: *Advanced Linux Programming*. New Riders Publishing, první vydání, 2001, ISBN 0-7357-1043-0.

[12] Mohring, M.; Vogdt, L.; Schröter, A.; aj.: *Build Service*. [online], [rev. 2008-12-28], [cit. 2009-01-06].
URL http://en.opensuse.org/Build_Service

[13] Shalloway, A.; Trott, J. R.: *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Addison Wesley, 2001, ISBN 0-201-71594-5.

[14] Slee, M.; Agarwal, A.; Kwiatowski, M.: *Thrift: Scalable Corss-Language Services Implementation*.
URL http://incubator.apache.org/thrift/static/thrift-20070401.pdf

[15] Wallengren, E.: *Continuous Automated Build and Integration Environment*. [online], [rev. 2008-01-06], [cit. 2009-01-06].
URL http://cabie.tigris.org/

Dodatek A

Obsah CD

Priložené CD obsahuje

- adresár `doc` – dokumentácia
- adresár `php` – tento adresár obsahuje web rozhranie projektu.
- adresáre `include`, `src` a `tinyxml` – tieto adresáre obsahujú zdrojové kódy projektu.
- `config.xml.default` – vzorový konfiguračný súbor s vysvetlením jednotlivých volieb. Pri inštalácii sa prekopíruje do adresára `/etc/cservice`.
- `INSTALL` a `INSTALL.sk` – pokyny na inštaláciu v anglickom a slovenskom jazyku.
- `README` – popis obsahu CD a jeho použitie.
- `db.sql` a `fk.sql` – SQL skripty na vytvorenie čistej databázy.
- `rpc.thrift` – definícia RPC protokolu.
- `CMakeLists.txt` – konfigurácia prekladového systému *CMake*.
- adresár `test` – obsahuje distribúciu projektu TinyXML, ktorú možno použiť na otestovanie funkčnosti implementácie.
- adresár `test-bezp` – obsahuje zdrojové kódy programu, ktorý sa pokúša prelomiť ochranu.
- `EUPL Licencia.pdf` – Plné znenie EUPL licencie, pod ktorou je šírený tento projekt.
- `EUPL Licencia.sk.pdf` – Oficiálny slovenský preklad EUPL licencie, pod ktorou je šírený tento projekt.
- `EUPL Licencia.cz.pdf` – Oficiálny český preklad EUPL licencie, pod ktorou je šírený tento projekt.