VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNCHRONIZACE A ZÁLOHOVÁNÍ DAT POD OS LINUX

SYNCHRONIZATION AND BACKUP OF DATA UNDER OS LINUX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ADAM MARŤÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL OČENÁŠEK, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Bakalářská práce se věnuje synchronizaci a zálohování dat. Cílem práce bylo navrhnout a implementovat vlastní nástroj pro synchronizaci a zálohování dat pod OS Linux. Realizace řešení byla inspirována programem Git. Pro realizaci byla použita architektura peer–to–peer. Pro ukládání metadat a vlastních dat nástroj používá databázi. Na základě testování bylo zjištěno, že nástroj má určitou režii při synchronizaci a zálohování dat. V závěru práce jsou tyto výsledky diskutovány a jsou předloženy další rozšiřující návrhy.

Abstract

This bachelor's thesis deals with synchronisation and backup of data under Linux OS. The aim of this paper is to design and implement a custom tool for data synchronisation and backup for Linux OS. The solution was inspired by the program Git. The architecture peer–to–peer was used to implement the project. The tool uses a database to store metadata and data itself. Based on testing, it was found that the tool has a certain overhead during the data synchronisation and the process of backing up. These results are discussed and other proposals are submited in the conclusion.

Klíčová slova

Zálohování, synchronizace, data, Linux, Boost, SQLite, algoritmus rsync, peer, síťová komunikace.

Keywords

Backup, synchronization, data, Linux, Boost, SQLite, rsync algorithm, peer, network communication.

Citace

Adam Marťák: Synchronizace a zálohování dat pod OS Linux, bakalářská práce, Brno, FIT VUT v Brně, 2013

Synchronizace a zálohování dat pod OS Linux

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavla Očenáška, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adam Marťák
13. května 2013

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Pavlovi Očenáškovi Ph.D. za jeho cenné připomínky a odborné vedení mé práce.

Obsah

Kapitola 1

Úvod

Vzhledem k současnému nárůstu informačních technologií a postupného rozšiřování elektronických dokumentů i dat, stoupá potřeba sdílení mezi různými platformami a zálohování těchto dat. Zálohování dat je proces, kdy se vytváří kopie dat, aby se zamezilo riziku ztráty. K zálohovaným datům je možné se kdykoliv vrátit. Pokud dojde ke ztrátě či neúmyslné změně, může nám to v budoucnu při práci s těmito daty ušetřit čas. Synchronizace a sdílení dat je potřeba tam, kde se pracuje s různými technologiemi, ba i platformně odlišnými. Dále při práci ve skupinách, kde dochází k častému sdílení společně využívaných dat. V obou případech lze tyto činnosti automatizovat pro pohodlí uživatele, což může v důsledku také šetřit jeho čas.

Vzniklo tak mnoho nástrojů, které řeší zálohování či synchronizaci dat nebo obojí. Mezi nejznámější lze zařadit Dropbox, Google Drive či Microsoft SkyDrive. Každý z těchto programů řeší danou problematiku podobným způsobem, existují však určité odlišnosti, které budou dále v textu rozvedeny. Většina těchto služeb nabízí také Cloud computing[1], což v případě některých uživatelů či firem, které mají své technologické a softwarové řešení, nemusí být výhodou z hlediska dostupnosti, ceny či licenčních podmínek k použité službě. Oproti tomu mezi výhody patří, že odpadá nutnost se starat o úložiště z fyzického pohledu. Spousta těchto služeb nabízí také přístup k datům přes webové rozhraní a tím odpadá nutnost mít pro sdílení speciální program. Soubor si tak může uživatel stáhnout přes webový prohlížeč. Ke službám sdílení a zálohování dat jsou nabízeny mnohdy další služby pro spolupráci mezi uživateli přes webové rozhraní. Mezi ně patří editace a prohlížení elektronických dokumentů v reálném čase.

Tato práce si klade za cíl navrhnout a implementovat nástroj, který bude synchronizovat a zálohovat data pod operačním systémem Linux. Nástroj by měl podporovat obecný protokol pro výměnu dat, aby bylo možné synchronizovat a zálohovat data mezi různými platformami. V dnešní době je určitě nutnost se zabývat bezpečností přenášených dat tak, aby se tato data nedostala k neoprávněným uživatelům, kteří by je mohli zneužít ve svůj prospěch. Proto bude při návrhu tohoto nástroje kladen důraz na bezpečnost přenosu a uchování dat. Dalším důležitým faktorem pro návrh tohoto nástroje je efektivita. Ta by v důsledku mohla snížit nároky na zdroje jako třeba využití přenosového pásma použité sítě a snížit potřebný prostor na disku pro zálohu dat.

[1] Cloud computing nabízí využívaní hardwarových a softwarových zdrojů dostupných přes Internet.

Kapitola 2

Synchronizace a zálohování dat

V této kapitole budou popsány obecné principy synchronizace a zálohování dat z různých hledisek. Nejprve bude vysvětleno, co to synchronizace a zálohování dat je. Rozdíl mezi nimi je ten, že synchronizace umožňuje sdílení dat v prostoru a zálohování umožňuje ukládaní dat v průběhu času [6]. Synchronizace dat je tedy proces sdílení dat mezi několika místy. To může přinášet problémy s konzistencí těchto dat, které je nutné řešit. Častým problémem může být vznik konfliktu mezi stejnými daty, které jsou upraveny různými uživateli na různých místech ve stejnou dobu. Taková data potom nemusí být jednoduché synchronizovat. Naproti tomu zálohování dat je proces, kdy se data periodicky ukládají a tím se zamezuje jejich ztrátě. Zálohovaná data pak v časové linii vytvoří historii verzí těchto dat. S pojmem zálohování dat také neodmyslitelně souvisí obnova dat. Obnova dat je zpětný proces oproti zálohování. Souvisí to také s pojmem historie verzí dat, která nás informuje o tom, z jakých dat je možné obnovu provést.

2.1 Synchronizace dat

Synchronizace dat nám má v ideálním případě přinést sdílení dat mezi různými místy [6]. Tento proces nezajišťuje zálohování dat, jelikož nepřináší možnost obnovy dat, ale pouze duplikaci. Přínosem takového sdílení je, že na všech místech může kdokoli pracovat s těmito daty nezávisle na sobě.

Při synchronizaci se musí zajistit, aby neaktuální data byla synchronizována s pozměněnou formou stejných dat z jiného místa. To přináší problém konfliktu, pokud budou stejná data pozměněna i na jiných místech. Tyto konflikty se dají řešit různými způsoby a záleží jen na uživateli a nástroji, který pro synchronizaci používá a jaké možnosti pro řešení konfliktů se mu nabízí.

2.2 Zálohování dat

Zálohování dat je kopírování určených dat v určitém čase do určitého místa [22]. Pro zálohování lze použít dva základní koncepty. Záloha je buď provedena tak, že se kopírují vždy všechny soubory nebo pouze změny těchto souborů [7]. Zálohování změn souborů se dá ještě rozlišit na inkrementální a diferenciální zálohování. Oba typy zálohování se kombinují s úplnou zálohou a jsou vhodné tam, kde se data, která jsou zálohována, příliš nemění [22].

Opačným procesem k zálohování je obnova dat.

2.2.1 Úplné zálohování

Je nejjednodušší technika pro zálohování. Zálohují se všechna data bez ohledu na to, zda se změnila od poslední zálohy [7]. Pokud je třeba data obnovit, jsou ve stejné podobě jako při záloze. Tento typ zálohy obsahuje vždy všechna zálohovaná data. To ovšem nese také nevýhody, jako velikost zálohy a s tím související délka procesu samotného zálohování. Pokud je nutné zálohovat velké objemy dat, bude pro zálohovaná data potřeba několikrát většího úložiště závislého na velikosti zálohovaných dat. Zde záleží na tom, zda se bude provádět úplná záloha pravidelně a bude uchována po určitou dobu nebo se provádí pouze jedna záloha za určitý čas a při další záloze se původní odstraní. Pokud je nutné obnovit soubory, které se změnily nebo porušily před několika zálohami, nebude už možnost je obnovit, jelikož poslední záloha bude obsahovat již jen tyto chybné soubory. Při tomto typu zálohování by bylo řešením mít tedy několik úplných záloh, to je ale, jak už bylo řečeno, náročné na paměť. Pro takovéto případy by bylo vhodnější zálohování formou změn oproti jedné úplné záloze.

2.2.2 Inkrementální zálohování

Při inkrementálním zálohování se uchovávají jen ta data, která se relativně změnila od poslední zálohy [22][7]. Na začátku cyklu takového zálohování se provede prakticky úplná záloha. Inkrementální zálohování pak vede k šetření paměti zálohovacího média.

Při tomto typu zálohování je nutné zjišťovat, která data se změnila. Na různých platformách je tento problém řešen jiným způsobem. Všechna data mají svá metadata. Na platformě Windows je v metadatech souboru příznak archivní bit, který by měl signalizovat, zda se data od posledního zálohování změnila. Při každé změně či vytvoření souboru je tento bit nastaven tak, že byla provedena změna. Při zálohování je pak tento bit nastaven tak, že změna nebyla provedena. Toto může způsobovat problémy v případě vícero programů pracujících se změnou tohoto bitu, jelikož se potom některá data, ač se od poslední zálohy změnila, nemusí zálohovat [24]. Jiná možnost detekce změny je pomocí času poslední změny dat nebo metadat souboru. Tento systém se používá hlavně na unixových systémech.

Obnova takových dat je složitější než v případě úplné zálohy. Jelikož jsou zálohy na sobě závislé, pro obnovu určitých dat v určitém čase je potřeba úplné zálohy a všechny inkrementální zálohy do tohoto času. Pokud dojde k poškození dat některé z inkrementálních záloh, neexistuje možnost obnovit zálohovaná data od poškozené zálohy.

Alternativou pro inkrementální zálohování může být úrovňové zálohování [22]. Každá záloha má svou úroveň a přiřazené pořadové číslo podle toho, kdy byla vytvořena. První záloha v cyklu je úplná a má úroveň 0. Následující záloha má úroveň 1 a obsahuje pouze změny od poslední zálohy. Další záloha bude úrovně 2 a obsahuje změny relativně k úrovni 1 atd. Po určitém čase se cyklus opakuje.

2.2.3 Diferenciální zálohování

Při diferenciálním zálohování se uchovávají soubory změněné vždy od první úplné zálohy [22]. Je to výhoda oproti inkrementálnímu zálohování. Pokud by se poškodila některá z diferenciálních záloh, tak budou ztracena data pouze v této záloze. Nevýhodou může být to, že po více procesech zálohování nebo při velkých změnách zálohovaných souborů mohou zálohy dosáhnout prakticky stejné velikosti jako úplná záloha.

Při obnově dat je pak zapotřebí úplné zálohy a diferenciální zálohy z času, kdy je nutné data obnovit.

2.3 Systémy správy verzí

Tyto systémy nám umožňují spolupráci mezi různými účastníky a uchovávají stav souborového systému (nejčastěji ve formě stromu) jak se vyvíjel v čase [5]. Uživatelé mají pak možnost se kdykoliv vrátit k předchozím stavům souborového systému. Důležitou vlastností je řešení sdílení souborů a jejich konfliktů mezi uživateli [23]. Základem takovýchto systémů je repozitář, který obsahuje právě zmíněné různé verze souborů a další informace v závislosti na konkrétním systému.

Jedna z možností, jak může uživatel verzovat soubory, je lokální správa verzí [5]. Tato forma je použitelná pouze pro jednoho uživatele. Proto existuje další forma, tzv. centralizovaný systém správy verzí [23]. Ta umožňuje práci více uživatelů najednou. Centralizace nám umožňuje mít celou práci na jednom místě a uživatelé si stahují pouze aktuální verzi souborů. Pokud je třeba obnovit starší verzi souboru, stáhne si ze serveru požadovanou verzi. Nevýhodou může být, že je systém právě na jednom místě. V případě výpadku nemá uživatel možnost pracovat se vzdáleným systémem, kromě své lokální verze. Mezi představitele patří např. programy CVS[1] a Subversion[2] [23].

Dalším typem jsou distribuované systémy správy verzí [5]. Tyto systémy uchovávají obsah celého repozitáře na straně uživatele. Uživatel nemá tedy problémy jako u centralizovaného systému a může s repozitářem pracovat prakticky pořád. Pokud repozitář obsahuje velké množství souborů a verzí souborů, může být náročný pro lokální paměťové médium. Nejznámějšími představiteli jsou Git[3], Mercurial[4] či Bazzar[5].

[1] Nástroj Concurrent Versions System je dostupný na adrese `http://cvs.nongnu.org/`.
[2] Nástroj je dostupný na adrese `https://subversion.apache.org/`.
[3] Nástroj je dostupný na adrese `http://git-scm.com/`.
[4] Nástroj je dostupný na adrese `http://mercurial.selenic.com/`.
[5] Nástroj je dostupný na adrese `http://bazaar.canonical.com/`.

Kapitola 3

Rešerše existujících programů

Tato kapitola popisuje existující řešení programů pro synchronizaci nebo zálohu dat a porovnává tyto programy z technického pohledu. Existuje celá řada programů pro tyto účely. Pro srovnání zde budou uvedeny jedny z těch nejpoužívanějších řešení s ohledem na použití pro návrh a implementaci vlastního programu. Dále zde budou popsány jednoduché programy pro synchronizaci jako je csync[2] [1], rsync a jeho vlastnosti pro efektivní přenos souborů po síti. V poslední řadě zde bude popsán nástroj pro správu verzí Git.

3.1 Google Drive

Google Drive je jedna ze služeb pro sdílení souborů. Používá architekturu klient-server. Tato služba je propojena s dalšími službami od Google. Disk Google [12] je k dispozici pro následující platformy:

- PC (Windows) a Mac,
- Chrome OS,
- iPhone a iPad,
- Zařízení se systémem Android.

Přístup je možný také přes webový prohlížeč. Google Drive používá volně šiřitelný software jako SQLite, pySqlite2, Python 2.6[2] a další. Google Drive je nabízen bezplatně do velikosti úložiště 5GB. Větší úložiště než 5GB jsou placené.

3.1.1 Platforma Linux

Pro platformu Linux neexistuje oficiální klient od Google. Google zveřejnil API[3] [13] k této službě a pro Linux vznikl neoficiální klient Grive[4]. Tento klient nenabízí zdaleka takové funkce jako oficiální klient pro jiné platformy, nelze propojovat službu s jinými službami Google jako jsou třeba Google dokumenty. Má také omezení ve sdílení souborů, konkrétně sdílení jednoho stejného souboru, který může existovat v různých adresářích, jelikož Google Drive uchovává pouze jednu kopii takového souboru.

[1] Nástroj lze nalézt na http://oss.linbit.com/csync2/.
[2] Tento skriptovací jazyk lze nalézt na http://www.python.org/.
[3] Zkratka označuje rozhraní pro programování aplikací.

3.1.2 Platforma Windows

Pro stažení a používání oficiálního klienta je nutné mít Google účet. Po stvrzení licenčních podmínek, instalaci a přihlášení do účtu Google v Drive klientu se vytvoří v systému složka se všemi sdílenými soubory. Pokud jsou roztříděny soubory do složek, lze si vybrat, které z těchto složek se mají synchronizovat. Tato možnost lze vybrat v nastavení. U každé složky a souboru se v průzkumníku zobrazuje ikonka s aktuálním stavem, ta může být buď synchronizace, synchronizováno nebo ještě nesynchronizováno. Zobrazování těchto ikon lze volitelně změnit v nastavení. Soubor či celá složka se synchronizuje ihned po vytvoření či změně. Synchronizace lze ručně pozastavit, v takovém případě se soubory a složky okamžitě nesynchronizují. Se synchronizací lze opět pokračovat po ručním obnovení v démonu. Pokud vznikne konflikt v souboru se stejným názvem, tak se řeší následovně. V systému, kde vznikl konflikt, vytvoří konfliktní soubor se stejným názvem a za něj přidá sufix v hranatých závorkách se slovem "konflikt"s původní koncovkou. Tento soubor potom synchronizuje.

3.1.3 Google Drive API

Google nabízí k této službě API [13]. Službu Google Drive lze tady jednoduše integrovat do dalších aplikací. Pro integraci je dostupný velký výběr programovacích jazyků. API používá HTTP[4] protokol. Pro bezpečnou komunikaci používá TLS[5]. API reference je roztříděna podle typu zdrojů, každý má různé metody a vrací svá data. Typy zdrojů, se kterými Google Drive pracuje:

- Soubory (Files) - získání metadat souborů a manipulace se soubory.
- Informace (About) - získání informací o účtu.
- Změny (Changes) - získání změn souborů.
- Položky (Children) - práce s položkami v adresáři.
- Nadřazené položky (Parents) - práce s adresáři.
- Oprávnění (Permissions) - získání a změna práv.
- Verze (Revisions) - práce s verzemi souborů.
- Aplikace (Apps) - programy uživatele, které podporují a pracují se soubory.
- Komentáře (Comments) - komentáře souborů.
- Odpovědi na komentáře(Replies) - odpovědi pro komentář.

Autentizace probíhá pomocí OAuth 2.0[6], což je sdílený systém přihlašování do služeb Google a jiných. Tímto odpadá nutnost se autentizovat uživatelským jménem a heslem při každém přístupu. Po první autentizaci se získá dlouhodobý "refresh token", který se uloží a použije pro autentizaci při dalším připojení.

Nahrávat soubory lze do velikosti 10GB s jakýmkoli typem média. Existuje několik možností jak nahrávat soubory:

[4]Hypertext Transfer Protocol je internetový protokol pro přenos hypertextových dokumentů.
[5]Transport Layer Security je následník zavrženého kryptografického protokolu Secure Sockets Layer.
[6]Specifikaci protokolu lze nalézt na http://oauth.net/2/.

- Jednoduché (simple upload) - pro malé soubory (soubor do 5MB).
- Vícedílné (multipart upload) - pro rychlý upload malých souborů a metadat.
- Obnovitelné (resumable upload) - umožňuje spolehlivý přenos pro velké soubory, možnost přenést i metadata, přenášet lze určené části souboru.

Lze stáhnout strukturovaný seznam všech souborů s metadaty. Stahovaní souborů probíhá přes získané identifikační číslo z vyžádaného seznamu všech metadat. Soubory lze stahovat i po částech. Lze stahovat i Google Dokumenty a převádět do jiných formátů. Soubory lze také stahovat přes vygenerovaný odkaz pro prohlížeč, který lze stáhnout mimo API. U souborů lze přímo měnit jejich metadata. Pokud není přidána koncovka v názvu, API ji samo přidá podle typu média. Google Drive automaticky indexuje soubory pro vyhledávání, pokud rozpozná typ média. Google Drive automaticky generuje miniatury souborů podle typu média.

Adresář má přesně určen speciální typ média jako *application/vnd.google-apps.folder* a nemá koncovku.

Google Drive automaticky vytváří historii revizí ke každému souboru. Přes API lze ukládat nové revize souboru nebo je možné si vyžádat historii revizí. Google Drive automaticky čistí revize s cílem optimalizace využití disku. Tato možnost se dá vypnout v nastavení. Lze stahovat obsah souborů jednotlivých revizí. Revize lze exportovat do podporovaných formátů a také je lze publikovat na web.

Aplikace používající Google Drive, které potřebují sledovat změny v souborech, by se musely opakovaně dotazovat změn, což je neefektivní a náročné na zdroje. Přehled změn si lze vyžádat pro všechny soubory najednou. Přehled poskytuje současný stav každého souboru, a to pouze v případě, že se soubor změnil od posledního "changestamp". Changestamp je unikátní, monotónně rostoucí číslo reprezentující změnu uživatelova souboru. Pro zjištění budoucích změn je třeba si toto číslo uložit. Tento systém zajišťuje, že aplikace nedostane stejnou změnu dvakrát.

Soubory lze vyhledávat nebo filtrovat podle různých kritérií.

3.1.4 Srovnání existujících služeb

Jednoduché srovnání s ostatními podobnými službami je uvedeno v tabulce 3.1. Všechny tyto služby jsou klient–server řešení a jsou přístupné přes web. V tabulce je vidět, že rozdíly jsou pouze v parametrech a speciálních službách k úložišti navíc. Spousta služeb je plně funkčních až po přechodu na placenou verzi.

3.2 Synchronizační nástroj *rsync*

Nástroj *rsync*[7] umožňuje synchronizovat soubory a adresáře mezi dvěma různými místy. K přenosu se používá delta algoritmu pro zmenšení objemu přenášených dat [27][28]. Výchozí možností pro kontrolu změn je kontrola atributů souborů a adresářů, konkrétně čas změny a jeho velikost. Lze také nastavit, aby detekoval soubory pouze se změnou velikosti souborů. Pomalejší metodou je kontrola pomocí součtu obsahu souborů.

Přednost a efektivita toho algoritmu spočívá v myšlence, že pokud jsou si dva soubory na různých místech podobné, je možné posílat pouze změny mezi těmito soubory. Tím se ušetří přenosové pásmo, takže lze algoritmus používat i na pomalých komunikačních

[7] Program lze nalézt na https://rsync.samba.org/.

	Google Drive	**Dropbox**	**SkyDrive**
platformy	Win, Mac, Chrome OS, iOS, Android	Win, Mac, Linux, BlackBarry, Android, Kindle Fire	Win, Mac, iOS, Windows Phone, Android
úložiště zdarma	5GB	2GB	7GB (dříve 25GB)
další služby	online dokumenty	-	online Office
dostupnost API	ano	ne	ano
historie verzí	všechny soubory	max 30 dní pro účet zdarma, placený neomezeně	pouze u online dokumentů

Tabulka 3.1: Tabulka srovnání serverových řešení

linkách. Problémem je, jakým způsobem se získávají změny mezi těmito soubory, když oba soubory jsou na jiných místech. Použil se tedy algoritmus, kdy se soubor rozdělí na stejně velké části, pro každou část se vypočtou dva otisky a pošlou se po síti druhé straně. Ta na základě této znalosti porovná postupně svůj soubor a zjistí, které části souboru jsou podle otisků shodné. Pokud nedojde v některém případě ke shodě, druhé straně se pošlou změny, které byly zjištěny. Druhá strana na základě těchto znalostí může sestavit soubor první strany. Zjišťování těchto změn probíhá pomocí dvou otisků, slabého a silnějšího, které musí jít rychle vypočítat. Porovnávají se nejprve slabé otisky, a v případě shody se porovnávají i silnější otisky. Pokud by se některé bloky neshodovaly již při slabém otisku, není třeba počítat a porovnávat silnější otisk.

3.3 Synchronizační nástroj *csync*[2]

Nástroj *csync*[2] umožňuje asynchronní synchronizaci souborů v počítačových clusterech. Nástroj umí řešit konflikty jednoduchým způsobem podle zadaných kritérií, například zda je soubor větší či menší nebo starší či novější. Ve výchozím nastavení není řešení konfliktů zapnuto. [29]

Pro zjišťování souborů, které se změnily, nástroj používá databázi metadat souborů. Ty pak porovnává se skutečnými soubory, které se mají synchronizovat. Pro metadata je použito databáze SQLite. Nástroj provádí změny vždy jen na dalších zařízeních v clustru. Pro zabezpečení komunikace je použito SSL, které lze vypnout a pro efektivní přenos dat se používá delta algoritmu rsync, implementovaný knihovnou *librsync*. Autentizace je řešena kombinací IP[8] adresy uzlů, certifikátem podepsaný sebou samým každého uzlu a sdíleného klíče. Uzly si vymění certifikáty při první komunikaci a uloží do databáze. [29]

Nástroj podporuje synchronizaci skupin uzlů, mezi kterými se mají soubory synchronizovat. Každá skupina potom obsahuje výrazy, kterými zahrneme nebo vyloučíme soubory, podle kterých se synchronizace řídí. Ke každé skupině může být přiřazeno neomezeně akcí, které budou provedeny po synchronizaci na vybraných souborech. Soubory, které budou při synchronizaci změněny, lze zálohovat do určeného adresáře. U souborů jsou synchronizovány oprávnění. [29]

[8]Internet Protocol je nejrozšířenějším protokolem v počítačových sítích, který pracuje na síťové vrstvě.

3.4 Git

Tento distribuovaný systém správy verzí byl navržen pro rychlost a jednoduchost s ohledem na velké projekty. Od ostatních systémů správy verzí jako třeba SVN[9] se liší způsobem ukládání dat a to ukládáním pouze souborů, které se změnily. Git ukládá vždy stav všech objektů, i když se nezměnily, v takovém případě uloží pouze odkaz na předchozí verzi [5].

Veškerá data repozitáře jako historie, logy a verze souborů se ukládají do skrytého adresáře *.git* v adresáři repozitáře. Aby bylo možné sdílet repozitář mezi všemi uživateli, nejlepším řešením je společný vzdálený repozitář na serveru [5]. Tento vzdálený repozitář má pouze adresář *.git* bez aktuálního pracovního adresáře. Pro přístup k repozitáři je možno použít různých protokolů jako třeba SSH[10], Git, HTTPS.

Git používá struktury jako objektový sklad a index. Všechny objekty ukládá a komprimuje pomocí zlib[11] do balíčku (packfile). Index popisuje, kde objekty v balíčku nalezne. Pro šetření místa Git nalezne všechny soubory se stejným názvem a podobnou velikostí. Mezi těmito podobnými soubory uloží pouze změny. Pro zrychlení přístupu k souborům ponechává novější soubory kompletní, u starších souborů ukládá pouze změny vůči těm novějším. [19][5]

V objektovém skladu mohou být uloženy čtyři atomické typy objektů, které jsou komprimovány. Jedním z nich je typ *blob* pro uchování dat souborů bez metadat. Dalším typem je *tree*, který jakoby reprezentuje obsah adresáře. Obsahuje metadata souborů a odkazy na samotná data, které jsou reprezentovány předchozím typem a rekurzivně odkazuje další objekty *tree*, čímž vytváří hierarchii dat. Objekt typu *commit* obsahuje metadata pro každou vytvořenou verzi, odkazuje se na objekty *tree*. Dalším typem je *tag*, který páruje objekt *commit* s uživatelovým názvem pro tuto akci. Všechny tyto objekty v objektovém skladu jsou identifikovány pomocí unikátního SHA-1[12] otisku samotného objektu [19][5]. Změna v objektu implikuje změnu SHA-1 otisku, ten se uloží jako nový objekt s názvem nového otisku.

[9] Zkratka představuje nástroj Subversion.

[10] Úplný název protokolu je Secure Shell.

[11] Je svobodná otevřená knihovna pro kompresi dat, dostupná na adrese `http://zlib.net/`.

[12] Secure Hash Algorithm je hašovací funkce, která vytváří ze vstupních dat otisk určité délky podle použité varianty algoritmu.

Kapitola 4

Možnosti pro synchronizaci a zálohování pod OS Linux

V této kapitole je popsána obecně platforma OS Linux a možnosti, které nabízí pro synchronizaci a zálohování dat. Nejprve bude popsáno, co to je Linux. Dále bude popsáno, jakou efektivní cestou se dají zjišťovat změny souborového systému a k čemu je to vhodné. Dále také budou popsány konkrétní systémy sledování změn souborového systému pro Linux. Samotná data a zálohy, které vznikají při přenosech, se musí efektivně uchovat. Budou zde popsány možnosti, které nám tato platforma nabízí. V dalších podkapitolách budou popsány techniky pro prezentaci dat při přenosu a samotné zabezpečení takového přenosu.

4.1 O platformě

Linux je robustní a plně použitelný svobodný operační systém unixového typu [25]. Jeho základem je Linuxové jádro. Celý systém je často označován jako GNU/Linux, kde zkratka GNU[1] představuje softwarovou výbavu pro běh a další vývoj. GNU/Linux je tedy chápán jako softwarové prostředí, které je poháněno právě Linuxovým jádrem. Linux nabízí programovací rozhraní původního Unixu. Jelikož by takový systém bylo složité sestavit ručně, vznikly tak mnohé distribuce, které se sestavením a samotnou distribucí sestaveného softwaru zabývají. Samotné archivy s binárními soubory se nazývají balíčky. Každá distribuce má většinou svůj balíčkovací systém, ale není to pravidlem. Spousty distribucí vychází z jiných distribucí a některé vlastnosti tak mohou mezi sebou sdílet.

4.2 Sledování událostí v souborovém systému

Aplikace, které pracují se souborovým systémem, většinou potřebují rychle reagovat na změny nebo obecně události v souborovém systému [20]. Proto by určitě nebylo vhodné, kdyby taková aplikace neustále prováděla čtení právě používaných adresářů a sama tak zjišťovala, co se změnilo či nezměnilo. Mohlo by dojít i k případům, kdy by bylo třeba pracovat s některým souborem a ten by ve výpisu souborů nemusel být nějakou dobu aktualizován nebo by již nemusel existovat.

Vznikl tak obecně systém pro monitorování událostí souborového systému. Na mnoha platformách existuje tento systém a většinou je nepřenositelný pro jiné platformy. V Linuxovém jádře se tento systém jmenuje *inotify*. Možné využití je třeba při výše zmíněných

[1] GNU je zkratka, která rekurzivně vyjadřuje "GNU není UNIX".

problémech, kdy se aplikace dozví okamžitě aktuální stav o adresáři či souboru, aniž by to musela sama pořád zjišťovat. Je možné tak sledovat třeba důležité soubory v systému a detekovat tak jejich změny či přístupy. Pro synchronizaci či zálohování to má také svůj přínos, jelikož tak můžeme okamžitě zjistit, které soubory máme synchronizovat nebo zálohovat [18].

4.2.1 Subsystém *inotify*

Inotify je subsystém Linuxového jádra a byl představen v jádře verze 2.6.13. Jeho předchůdce byl systém *dnotify*, který měl podobné možnosti jako *inotify*, ale vykonával je technicky jinak a celý systém sledování souborového systému byl složitý. *Inotify* se tak stal lepším řešením daného problému [18].

Celý systém pracuje následujícím způsobem. Aplikace, která chce zjišťovat změny v souborovém systému, inicializuje *inotify* a uchová si deskriptor souboru pro další použití. Aplikace musí dále nastavit, které soubory či adresáře chce sledovat. Aplikace musí mít práva pro čtení těchto položek. Pokud je sledován adresář, změny se budou týkat položek v adresáři i samotného adresáře. Pokud je potřeba sledovat adresáře rekurzivně, musí se přidat sledování pro každý adresář samostatně. K této každé sledované položce se musí určit typy událostí, pro které bude aplikace informována. Pro každý takto sledovaný objekt dostane aplikace sledovací deskriptor, který je potřebný pro další operace, jako třeba odebrání položky. Pro změnu sledovače již sledovaného souboru stačí zavolat stejnou operaci s pozměněnými typy událostí jako pro vytvoření sledovače. Pokud chce aplikace dostávat zjišťované změny, zavolá odpovídající operaci, která je blokující. Pokud proběhne událost v souborovém systému, aplikace obdrží jednu nebo více struktur s popisem změn sledovaných objektů. Pokud již sledování není potřeba, je nutno odstranit sledování pomocí získaného deskriptoru souboru při inicializaci *inotify*, tímto se zruší i všechny definované sledovače objektů k tomuto deskriptoru.

Jelikož subsystém *inotify* spotřebovává paměť jádra, jeho činnost musí být limitována. Limity jsou především pro délku fronty událostí, dále pro počet instancí *inotify* na uživatele a počet sledovaných položek na jednu instanci.

4.2.2 Ostatní mechanizmy

FAM[2] a *Gamin*[3] jsou abstraktnější mechanizmy pro sledovaní souborového systému a lze je použít i na jiných platformách [18]. Používá systémy *inotify* nebo *dnotify* tam, kde jsou dostupné.

Multiplatformní knihovna *Qt*[4] nabízí od verze 4.2 rozhraní pro sledování souborů a adresářů. To zajišťuje třída `QFileSystemWatcher`, která sleduje definované objekty. Na Linuxu tento systém používá *inotify*.

Další možností je jednoduchá C knihovna zvaná *inotify-tools*[5]. Knihovna je pouze pro Linux a používá *inotify*.

[2] Knihovnu File Alteration Monitor lze nalézt na adrese `http://oss.sgi.com/projects/fam/`.
[3] Knihovna je dostupná na `http://people.gnome.org/~veillard/gamin/`.
[4] Qt toolkit je dostupný na adrese `https://qt-project.org/`.
[5] Knihovna je dostupná na adrese `https://github.com/rvoicilas/inotify-tools`.

4.3 Uchování dat

Pro zálohování či uchování starších verzí souborů je nezbytné data efektivně uchovávat. Dále je třeba uchovat metadata těchto dat a všechny informace potřebné pro proces další synchronizace. Jednou z možností by bylo data uchovávat přímo v souborovém systému, což by asi nebylo elegantní řešení kvůli jednoduchému narušení integrity těchto dat. Existuje spousta jednoduchých relačních i nerelačních databází. Jednou z jednoduchých relačních databází je SQLite.

SQLite pro uchování dat a konfigurace používá samostatný soubor. Pro práci s tímto souborem nepotřebuje serverovou část, ale pracuje s ním přímo sama aplikace přes knihovní rozhraní. Databáze částečně podporuje standard SQL92. SQLite podporuje standardní funkce databází jako jsou transakce, dále zápis binárních dat apod. Výhodou zůstává, že všechny informace a data by byly na jednom jediném místě. Taková databáze by se dala opět jednoduše zálohovat a migrovat. [2]

Pro uchování samotných souborů by bylo vhodnější uložení do některé z nerelačních databází. Některé z nich jako GDBM[6], Oracle Berkley DB[7] a LevelDB[8] jsou jednoduché databáze pro ukládání dat ve formě klíč–data a umožňují práci s těmito strukturami. Neumožňují již však vytváření složitějších struktur.

4.4 Síťová vrstva

Internetový protokol zajišťuje komunikaci v rámci Internetu a lokálních sítí [16]. Každý prvek, který chce komunikovat v rámci Internetu, je identifikován unikátní IP adresou v rámci globální sítě. Existují dvě verze tohoto protokolu IPv4 a IPv6. Linuxové jádro podporuje obě tyto verze.

IPv4 je první verze IP protokolu, která se masivně rozšířila. V posledních letech se potýkáme s problémem nedostatku IP adres, jelikož IPv4 má pouze 32 bitové adresy. V roce 2011 došlo k úplnému vyčerpání volných bloků adres na globální úrovni. Tento problém řeší novější protokol IPv6, který má již 128 bitové adresy. Problémem zůstává, že IPv6 se pomalu rozšiřuje.

Linux nabízí také jednu z translačních přechodových metod SIIT[9], která umožňuje naslouchat pro obě IP verze v jednom socketu. Všechny IPv4 adresy, které se budou chtít připojit k tomuto socketu budou přeloženy do speciální formy IPv6 adresy s prefixem ::ffff:0:0:0/96 zvaným IPv4-translated addresses. Výsledná forma takové adresy může být například "::ffff:0:127.0.0.1". Aby tento translační mechanizmus na Linuxu fungoval, musí být nastavena následující možnost na 0: */proc/sys/net/ipv6/bindv6only*.

V sítích existuje mechanizmus NAT[10], který umožňuje, aby všechna zařízení lokální sítě vystupovala navenek pod jedinou IP adresou. Každé zařízení má přitom vlastní privátní adresu a NAT musí každou komunikaci s venkovní sítí překládat. S tím přichází také problém pro síťové aplikace, které potřebují aktivně naslouchat. Nejjednodušším řešením by bylo porty ručně nastavit a NAT bude tato pravidla zohledňovat. Dalším jednoduchým řešením

[6] Projekt GNU dbm je dostupný na adrese `http://www.gnu.org.ua/software/gdbm/`.

[7] Databáze je dostupná na adrese `http://www.oracle.com/technetwork/products/berkeleydb/overview/index.html`.

[8] Databáze je dostupná na adrese `https://code.google.com/p/leveldb/`.

[9] Celým názvem Stateless IP/ICMP Translation.

[10] Celý název mechanizmu je Network address translation.

může být UPnP[11], které ale není bezpečné kvůli absenci autentizace klienta. UPnP musí podporovat zařízení, které adresy překládá a musí být zapnuto. Nastavení UPnP implementuje například knihovna *miniupnpc*[12].

4.5 Zabezpečení přenosu dat

Pro zabezpečení nad transportní vrstvou lze použít SSH a SSL[13]/TLS. Zabezpečení lze provést také na nižších vrstvách, to by však nebylo v režii synchronizačního nástroje. Všechny již vyjmenované použitelné mechanizmy budou nyní popsány.

4.5.1 SSH

SSH je protokol, který používá klient/server architekturu [3]. Zajišťuje transparentní šifrování komunikace obou stran, autentizaci a integritu dat. To nám zajišťuje, že data, která budou zabezpečena, nebudou změněna ani prozrazena. SSH se nejčastěji používá pro vzdálenou správu systémů a vykonávání vzdálených příkazů zadané klientem. SSH obsahuje dvě navzájem nekompatibilní verze. SSH-1, která je již zavržená a SSH-2, která se dnes používá. SSH poskytuje více typů autentizace. Nejbezpečnější je autentizace asymetrickým klíčem, který identifikuje uživatele. Další možností je autentizace heslem. SFTP[14] je subsystém SSH, jelikož samotné SSH neumí posílat soubory. SSH implementují například knihovny *cryptlib*[15], *libssh*[16] a *libssh2*[17].

4.5.2 SSL a TLS

SSL či jeho nástupce TLS přináší zabezpečení mezi klientem a serverem [8]. Ač TLS vychází z SSLv3, oba protokoly nejsou navzájem interoperabilní. Pro šifrování se používá dvou vrstev. První vrstva (TLS Record Protocol) nad transportní vrstvou zajišťuje soukromí a spolehlivost. Pro zajištění identity je použito symetrické nebo asymetrické kryptografie. Vyjednání klíčů, šifrovacího algoritmu a vzájemné autentizace zajišťuje druhá vrstva (TLS Handshake Protocol) zapouzdřená v první vrstvě.

TLS podporuje tři možné druhy autentizace. Autentizaci obou stran, tedy jak serveru tak klienta, pouze autentizaci serveru bez autentizace klienta a autentizace anonymní.

Implementaci těchto protokolů zajišťují knihovny jako *GnuTLS*[18], *OpenSSL*[19], *cryptlib* a další. Oba zabezpečovací protokoly zajišťují bezpečný přenos pouze v případě, že jejich implementace jsou správně použity. O těchto rizicích se můžeme dočíst v [11].

4.5.3 *cryptlib*

Po konzultaci mi byla doporučena knihovna *cryptlib*. Tato knihovna umožňuje jednoduché použití šifrování a autentizace [15]. Knihovna je multiplatformní a nabízí podporu

[11] Záměrem Universal Plug and Play je zjednodušení připojení a používání periferních zařízení.
[12] Knihovna je dostupná na adrese `http://miniupnp.tuxfamily.org/`.
[13] Secure Sockets Layer je již zavržený protokol pro bezpečnou síťovou komunikaci.
[14] Zkratka představuj Secure File Transfer Protocol.
[15] Knihovna je dostupná na adrese `http://www.cs.auckland.ac.nz/~pgut001/cryptlib/`.
[16] Knihovna je dostupná na adrese `http://www.libssh.org/`.
[17] Knihovna je dostupná na adrese `http://www.libssh2.org/`.
[18] Knihovna je dostupná na adrese `http://www.gnutls.org/`.
[19] Knihovna je dostupná na adrese `https://www.openssl.org/`.

různých programovacích jazyků. Knihovna podporuje PGP[20], SSL/TLS, SSH a další zabezpečení. Podporuje také správu certifikátů a validaci X.509/PKIX[21].

4.6 Prezentace přenášených dat

Pro přenos dat nám architektura nabízí více možností. Pro přenos textového zápisu protokolu lze použít například XML[22] či JSON[23]. Pro binární zápis protokolu lze použít ASN.1[24], Protocol Buffers[25] nebo Apache Trift[26].

4.6.1 ASN.1

ASN.1 je formální notace pro popis zpráv protokolu síťových aplikací [9]. Není vázáno na konkrétní architekturu ani programovací jazyk. ASN.1 spadá do prezentační vrstvy OSI[27] modelu. Před samotným posíláním musí být určena nebo ustavena pravidla pro kódování (DER[28], PER[29] nebo další). Tato vrstva představuje data tak, aby mohla být druhou komunikující stranou dekódována.

Pro abstraktní zápis se používá speciální syntaxe. Struktury mohou obsahovat jednoduché datové typy a z těchto struktur mohou vznikat složené datové typy. Takto abstraktně zapsaná data jsou předložena kompilátoru, který vytvoří konkrétní struktury pro použitý programovací jazyk a funkce pro převod těchto struktur. Implementaci ASN.1 zajišťuje knihovna *libtasn1*[30].

4.6.2 Protocol Buffers

Protocol Buffers je jedna z dalších možností, jak přenášet datové struktury po síti v binární podobě. Knihovna je multiplatformní a nabízí také celou řadu programovacích jazyků [14]. Zápis protokolu je v textové podobě, ta se převádí pomocí programu *protoc*, který je součástí knihovny. Výsledek takového převodu je vytvoření tříd pro práci s definovanými strukturami pro programovací jazyk. Třídy se pak používají podle zvyklostí daného jazyka.

Struktury podporují základní datové typy a výčet. Lze vytvářet také složitější struktury z již definovaných struktur. Nevýhodou zůstává absence složeného datového typu unie. Zato definované struktury mohou mít položky nepovinné. Ve struktuře lze také určit, které položky mohou obsahovat stejný vícenásobný obsah.

Protocol Buffers nabízí také optimalizaci pro různé případy užití. Lze tak použít i odlehčenou variantu samotné knihovny *protobuf-lite*, která nenabízí některé pokročilé funkce pro určité platformy [14]. Pro výměnu dat po síti však bohatě dostačuje.

[20] Pretty Good Privacy je internetový standard pro šifrování a podepisování.
[21] Public-Key Infrastructure (X.509) je standard pro podepisování a správu certifikátů.
[22] Extensible Markup Language je obecný značkovací jazyk.
[23] JavaScript Object Notation je formát pro výměnu dat.
[24] Zkratka představuje Abstract Syntax Notation One.
[25] Knihovnu lze nalézt na adrese `https://developers.google.com/protocol-buffers/`.
[26] Knihovnu lze nalézt na adrese `https://thrift.apache.org/`.
[27] ISO/OSI představuje referenční model pro návrh síťových aplikací.
[28] Zkratka představuje Distinguished Encoding Rules.
[29] Zkratka představuje Packed Encoding Rules.
[30] Tato svobodná knihovna je dostupná na adrese `https://www.gnu.org/software/libtasn1/`.

4.6.3 Knihovna *librsync*

Knihovna *librsync*[31] implementuje delta algoritmus použitý v nástroji *rsync* popsaný v sekci 3.2. Tímto odpadá nutnost přenášet celý soubor, ale pouze části, které se změnily. To v důsledku může ušetřit čas synchronizace a přenosové pásmo.

[31] Knihovna je dostupná na adrese http://librsync.sourceforge.net/.

Kapitola 5

Návrh aplikace

Kapitola popisuje návrh systémového nástroje pro synchronizaci a zálohování dat. Důraz bude kladen na bezpečnost a protokol pro výměnu dat, aby bylo možné výměnu dat provádět s podobnými nástroji na jiných platformách. V další řadě je nutné, aby tyto činnosti probíhaly efektivně. Nejdříve budou definovány požadavky pro konkrétní synchronizační nástroj pro platformu Linux. Dále budou popsány modely síťové komunikace a s tím spojené problémy.

5.1 Požadavky na aplikaci

Cílem této práce je vytvořit synchronizační nástroj. Aplikace by měla splňovat tyto požadavky:

- spustitelné na OS Linux,
- decentralizovaná architektura,
- synchronizace dat mezi odlišnými platformami,
- zálohování dat - možnost obnovy starší verze souboru,
- zabezpečená komunikace, efektivita a bezpečnost celého systému,
- volitelné přívětivé uživatelské rozhraní.

Důvody pro zvolené vlastnosti, kromě zadaných, budou popsány dále v textu.

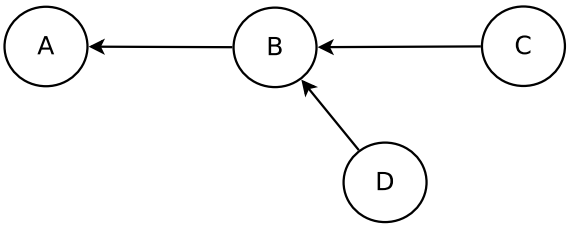
5.2 Síťová architektura

Pro návrh aplikace lze použít dva základní síťové koncepty, kterými jsou peer–to–peer a klient–server [21]. Klient–server architektura představuje jasně definovaný model pro komunikaci dvou stran, kdy klient se dotazuje serveru a ten mu odpovídá. V tomto modelu je jasně daná hierarchie. V modelu peer–to–peer jsou si při komunikaci všichni rovni a probíhající komunikace mezi uzly je rovnocenná. V tomto modelu mohou všichni komunikovat se všemi.

Pro návrh byl zvolen koncept peer–to–peer. Inspiraci, proč je tento model pro synchronizaci a zálohování přínosnější, můžeme hledat u programu Git. Každý uzel si uchovává celou historii včetně aktuální verze, každý uzel tak může používat tato data i v případě, že

jiný uzel není na síti zrovna dostupný nebo naopak. Distribuovanost také zajišťuje duplicitu dat pro případ jejich poškození. Pokud je potřebný uzel podobný serveru, jednoduše se na síti použije další distribuovaný uzel, ke kterému mají všechny ostatní uzly přístup přes veřejnou internetovou adresu.

Díky peer–to–peer architektuře může vzniknout topologie podobné topologii mesh. Na obrázku 5.1 lze vidět příklad této možné topologie. Peer A synchronizuje s uzlem B. Přes peer B synchronizují oba uzly C a D, oba budou synchronizovat nepřímo s uzlem A.



Obrázek 5.1: Ukázka možné synchronizační topologie

S modelem peer–to–peer by mohly nastat problémy při komunikaci přes Internet. Pokud by na obou stranách byl použit NAT, tak nebude možné iniciovat spojení. Tento problém a některá řešení byly popsány v sekci 4.4. V případě použití IPv6 spojení bez NATu by se tomuto problému dalo vyhnout, jelikož IPv6 disponuje větším adresním prostorem. Za předpokladu, že pro každý počítač existuje unikátní adresa, v případě IPv4 může nastat situace, kdy bude jedno zařízení za NATem a druhé bude mít globální adresu. Další možností je, že pokud budou oba uzly za NATem, kromě již zmíněných řešení se nabízí použít některou metodu NAT traversal, konkrétně techniku "hole punching"[10]. K tomu je ale potřeba třetího uzlu, který by musel být přístupný přes veřejnou internetovou adresu.

Pro komunikaci mezi dvěma uzly je potřeba určit jejich pořadí. Pokud se první uzel připojuje k druhému, bude první uzel klient. Je to z důvodu, že BSD[1] Sockety jsou navrženy pro architekturu klient–server. Při samotné komunikaci jsou si uzly již rovny. Aplikace bude podporovat obě verze IP protokolu vzhledem k postupnému rozšiřování IPv6. Pro návrh bylo použito referenčního modelu ISO/OSI, který lze vidět na obrázku 5.2.

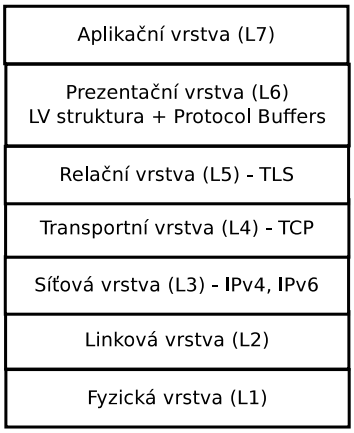
5.2.1 Bezpečnost

Bezpečnost přenosu lze řešit některou z TLS knihoven. Po konzultaci bylo doporučeno použít mechanizmus X.509. Autentizace zařízení by probíhala následujícím způsobem. Každé zařízení by vlastnilo svůj certifikát podepsaný sám sebou. V kapitole 4.5.2 byly uvedeny typy autentizace, které TLS podporuje. Pro náš případ je ideální použít autentizaci obou stran. Obě zařízení si navzájem zkontrolují certifikát a autentizují se.

5.2.2 Komunikační protokol

Pro popis protokolu bylo použito Protocol Buffers. Protokol byl navržen ve spolupráci se studenty z fakulty s podobným zaměřením bakalářské práce [17] [26]. Kompletní definici protokolu lze nalézt v příloze C. Tím je zaručeno, že je nástroj v rámci komunikace s podobnými nástroji na jiných platformách kompatibilní. Komunikaci může začít kdokoliv. Jelikož jsou si peery rovny, oba peery mohou komunikovat navzájem mezi sebou najednou.

[1] Zkratka představuje Berkeley Software Distribution.



Obrázek 5.2: Síťová architektura

Každá vytvořená zpráva v Protocol Buffers je zapouzdřena do length-value struktury podobná BER TLV[2] struktuře. Jelikož typy zpráv jsou obsaženy až ve struktuře Protocol Buffers, není nutné typ zprávy uvádět. Tím nám vznikne jednoduchá length-value struktura. První čtyři bajty length-value struktury představují velikost zprávy v Protocol Buffers. Dále následuje obsah samotné serializované zprávy.

Komunikace je rozdělena do zpráv typu oznámení, požadavek a odpověď. Každá zpráva typu požadavek nebo odpověď obsahuje transakční identifikátor. Na počátku komunikace si obě zařízení vymění povinnou zprávu o rozšířeních protokolu a představí vlastní identifikátor a název. Následuje výměna informací o svazcích. Po úvodní výměně informací začne komunikace požadavek–odpověď. Jeden požadavek může mít i více odpovědí. Požadavek může následovat další požadavek, pokud je to v předchozím požadavku specifikováno.

Pokud nástroj detekuje přítomnost některého z jeho svazků u vzdáleného uživatele, může začít se synchronizací. Nejprve si od druhé strany vyžádá identifikátor posledního známého snímku. Pokud vzdálený uživatel nemá zatím žádný snímek pro daný svazek, nebude žádný identifikátor snímku ve zprávě vyplňovat. Tím synchronizace skončí, jelikož není co synchronizovat.

Poté, co nástroj obdrží poslední snímek, zkontroluje svou vlastní databázi, zda již tento snímek obsahuje. Pokud byl snímek nalezen, tak databáze již obsahuje všechny změny druhé strany. Pokud snímek nebyl nalezen, tak nástroj pokračuje dotazem na tento snímek. Po přijetí této zprávy zjistí, zda má snímek rodiče. Pokud rodič existuje a nemá jej v lokální databázi, nástroj musí pokračovat v dotazování tak dlouho, dokud nenalezne poslední synchronizovaný snímek nebo dokud nenalezne snímek, který již nemá rodiče.

Po nalezení posledního nového snímku, který už není v lokální databázi, může začít dotazování na objekty metadat a samotných dat. Pokud nástroj stáhne všechny objekty vázané na tento snímek, může jej uložit do své databáze jako poslední snímek pod stejným identifikátorem a pokračuje na další nový snímek, který ještě nemá. Takto pokračuje dokud nesynchronizuje všechny vzdálené snímky. Po dokončení synchronizace nástroj pošle oznámení, že je synchronizován.

Při ukládání cizích snímků do vlastní databáze je důležité zachovat identifikátor snímku. To proto, aby nástroj, kterému původně snímek patřil, mohl detekovat vlastní snímky v cizí

[2]Zkratka představuje type-length-value.

databázi a nestahoval je tak znovu. Jinak by mohlo dojít k opětovnému stahování vlastních dat.

Při poslání zprávy pull lze specifikovat, zda má vzdálený uzel posílat změny, které se objeví postupně při práci s pracovním adresářem svazku. Posílání těchto změn lze kdykoliv zastavit jinou zprávou.

Pokud dojde při zjišťování odpovědi k chybě, nástroj má možnost odeslat chybovou zprávu společně s popisem chyby.

Pro efektivní přenos dat mezi uživateli by mělo být použito rsync delta algoritmu. Ten může být použit v případech, kdy jedna strana, která požaduje data určitého souboru, má k dispozici také dřívější verzi souboru. V jiných případech je nutné posílat vždy celý soubor. Použití tohoto algoritmu není povinné ani v případech, kdy by to bylo vhodné. Některá z komunikačních stran totiž nemusí toto rozšíření vůbec implementovat.

5.3 Uchování stavu a dat

Soubory a adresáře aktuálního pracovního adresáře se budou synchronizovat vždy podle definovaného svazku. Každý svazek má pevně danou cestu v systému a volitelný název. Historie verzí souborů se může uchovávat podle nastavení uživatele. Pro uchování historie lze použít některou z databází popsaných v sekci 4.3.

Dále je potřeba ukládat veškeré informace pro aktuální stav, všechna metadata k souborům a samotná data souborů. Každý uzel (peer), objekt a svazek bude identifikován pomocí UUID[3], což by mělo zaručit, že žádný z vyjmenovaných entit nebude mít stejné identifikační číslo. Objektem jsou data, metadata a snímek (snapshot). Snímek zde představuje novou verzi změn a může obsahovat jednu nebo více položek metadat.

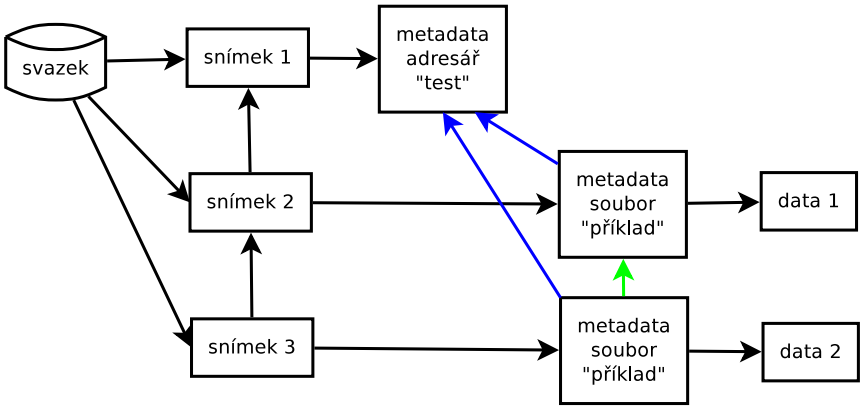
Na obrázku 5.3 je ukázka možné hierarchie objektů. Je zde jeden svazek s názvem *svazek*. V tomto svazku se postupně provedly tři změny, které představují tři snímky. Snímky jsou na sobě závislé, aby bylo možné určit posloupnost změn. Snímek 1 nemá žádného rodiče, je tedy první změnou. Snímek 1 ukazuje na objekt metadata typu adresář s názvem *test*. Metadata *test* nemají žádného přímého předka, to znamená, že byl adresář vytvořen. Jelikož adresář sám nemůže obsahovat data, nemůže odkazovat na objekt data. Dále zde je snímek 2, který ukazuje na metadata typu soubor s názvem *příklad*. Tato metadata náleží do adresáře *test*, to je vyjádřeno modrou šipkou. Metadata typu soubor mohou obsahovat data, což v našem příkladě představuje šipka na objekt data 1. Dále je zde snímek 3, který ukazuje na metadata typu soubor s názvem *příklad*. Tato metadata mají předka, což vyjadřuje zelená šipka a obsahují také vlastní data, což vyjadřuje šipka na objekt data 2.

Jiná možnost je identifikovat objekty pomocí otisku SHA-1, jak to dělá Git. To by na platformách s omezenými zdroji mohlo být náročné a neefektivní. Pro jednoduchou detekci, zda se data souboru změnila, by bylo možné použít SHA-1 otisk souboru. Tím by se dalo jednoduše detekovat, zda se data souboru změnila nebo ne. Pokud by se data nezměnila, vytvořil by se další nový objekt metadata s původními daty.

Pokud chceme uložit binární data v některé z databází, musíme při návrhu počítat s jejich limity. Všechny dříve uvedené databázové systémy mohou uložit maximálně 1 GB binárních dat do jednoho záznamu [2]. Pokud budou soubory větší než 1 GB, musí se data rozdělit do více záznamů.

Uložená data jsou vždy úplná. Nástroj tedy bude zálohovat vždy úplnou kopii dat jednotlivých dat. Pro efektivní ukládání dat by bylo možné ukládat jen změny mezi dvěma

[3] Zkratka představuje Universally unique identifier.



Obrázek 5.3: Ukázka hierarchie objektů

objekty typu data. To by mohlo být implementačně náročné, proto se budou data ukládat vždy v úplné formě.

5.3.1 Zjišťování změn souborů a adresářů

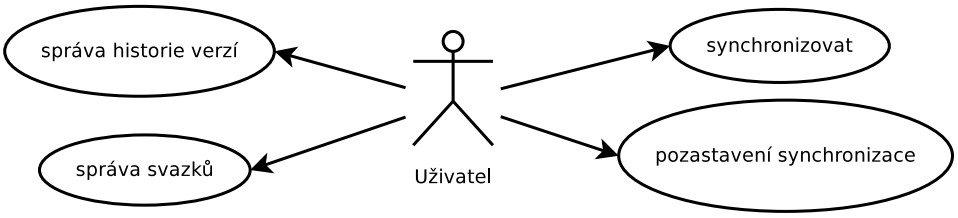
Pokud synchronizační nástroj poběží právě v systému, změny souborů, adresářů a jejich atributů může sledovat přes systém *inotify*. Na základě těchto změn může okamžitě vytvořit nový snímek a neprodleně o těchto změnách informovat další zařízení.

V případě, že nástroj nebude spuštěn, je nutné všechny změny efektivně detekovat při opětovném spuštění nástroje. Tohle lze řešit kontrolou atributů souboru, kterými jsou čas změny, velikost a čas změny i-uzlu. Podobně to provádí nástroj rsync. Proto je nutné, uchovávat i tyto atributy všech souborů a adresářů. Pro každý soubor a adresář je nutné provést tuto kontrolu a ověřit, zda se něco nezměnilo. Pokud by k nějaké změně došlo, všechny změny se uloží v novém snímku.

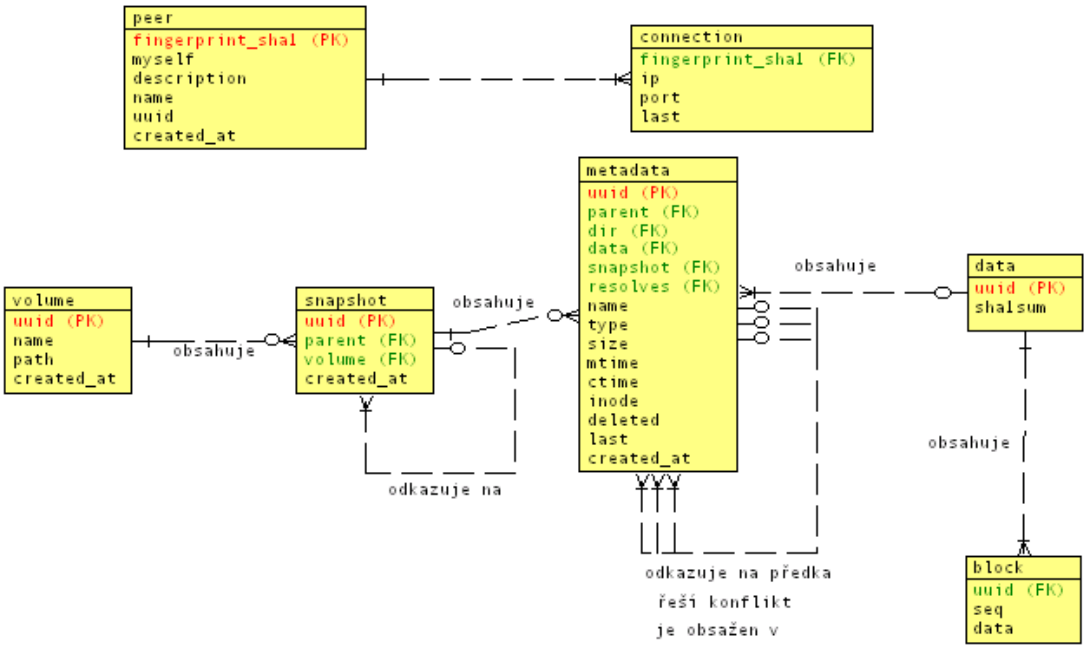
Pro efektivní detekování adresáře, který byl přejmenován, když nástroj nebyl zrovna spuštěn, lze využít i-uzlu. V případě přejmenování adresáře zůstává i-uzel stejný. Není tomu tak v případě změny obsahu souboru. Může však nastat také případ, kdy původní adresář s určitým i-uzlem byl smazán a byl vytvořen nový adresář či soubor, který dostal i-uzel předchozího adresáře. Proto je potřeba pro použití této detekce ověřit podobnost obsahu adresáře a určit tak, zda se jedná o ten samý adresář.

5.4 Analýza požadavků

Hlavní funkce je synchronizace svazků. Ta může být uživatelem pozastavena. Uživatel musí mít možnost spravovat svazky. Pokud si chce uživatel obnovit soubor ze zálohy, bude potřebovat správce verzí. Podle požadavků na nástroj byl sestaven diagram případů užití 5.4 a ER diagram 5.5 podle vztahů mezi jednotlivými objekty. Pro jednoduché nalezení posledních dat v databázi byl přidán atribut *last* do entity metadata. Entita *peer* zde představuje jednoho uživatele a může obsahovat jednoho lokálního uživatele. Další uživatelé budou vzdálení. Jeden uživatel se může připojovat z více adres.



Obrázek 5.4: Diagram případů užití



Obrázek 5.5: ER diagram

5.4.1 Spuštění nástroje

Po spuštění nástroje by mělo dojít k vygenerování certifikátu, kterým se bude uživatel dále autentizovat proti dalším uživatelům. Uživatel si zvolí jméno a vygeneruje si jedinečný identifikátor uživatele. Po prvním spuštění musí uživatel vytvořit alespoň jeden svazek, aby mohl synchronizovat. Poté se uživatel může připojit k dalším uživatelům a začít synchronizovat.

5.4.2 Správa svazků

Uživatel potřebuje svazky spravovat, proto musí existovat správce svazků. Každý svazek je identifikován jedinečným UUID. Při vytváření svazku musí zadat vlastní název svazku, ten se bude také zobrazovat dalším uživatelům. Dále je potřeba zadat cestu k pracovnímu adresáři svazku, která musí být jedinečná a nesmí být podmnožinou nebo nadmnožinou již existujícího uživatelova svazku.

Pokud uživatel chce vytvořit svazek na základě svazků ostatních uživatelů, musí se nejdříve připojit k jinému uživateli a zjistit, které vzdálené svazky jsou dostupné. Na zá-

kladě těchto informací si vytvoří lokální svazek s vlastní cestou k pracovnímu adresáři. Při vytvoření svazku se musí okamžitě začít zaznamenávat události v pracovním adresáři daného svazku.

Pokud nastane některá z událostí jako vytvoření souboru či adresáře, změna souboru, změna atributů souboru či adresáře, přejmenování adresáře a smazání souboru či adresáře, nástroj musí na tyto události neprodleně reagovat vytvořením nového snímku. Při každém dalším spuštění nástroje musí dojít k detekci změn, které se udály, když nástroj nebyl spuštěn.

5.4.3 Synchronizace

Uživatel započne synchronizaci tím, že iniciuje spojení s dalším uživatelem nebo sám se sebou na jiném zařízení. Při navázání nového spojení musí uživatel ověřit otisk certifikátu a potvrdit uložení otisku do databáze. Při dalším připojení už kontrola otisku certifikátu nebude třeba. Ověření provede i druhá strana, která si ověří také otisk certifikátu a uloží do své databáze.

V případě, že na obou zařízeních existuje svazek se stejným identifikátorem, synchronizace může začít. Pokud svazek neexistuje, musí byt vytvořen pomocí správce svazků. Pokud při připojení dvou uživatelů dojde k některé události v souborovém systému, nejprve ji musí nástroj sám zavést do vlastní databáze a poté informovat ostatní připojené uživatele.

Pokud uživatel z nějakého důvodu nechce pro určitý čas používat synchronizaci s ostatními, může synchronizaci jen pozastavit a nemusí ukončovat samotný nástroj. Tím se přeruší všechny právě probíhající synchronizace a až do opětovného spuštění synchronizace se nebudou propagovat změny dalším uzlům. Detekce změn v souborovém systému však nadále pokračuje a zálohují se soubory do lokální databáze.

5.4.4 Historie verzí

Správce historie slouží pro jednoduché sledování a obnovu souborů či adresářů ze vzniklé databáze. Uživatel by si měl pomocí správce vypsat všechny soubory a jejich verze, které databáze obsahuje. Uživatel by měl mít možnost jednoduché obnovy požadovaného objektu. Obnova souborů by však měla probíhat mimo pracovní adresář svazku, který si uživatel musí předem určit. Pokud chce poté soubor použít za aktuální, může tento soubor přesunout do pracovního adresáře svazku.

5.5 Uživatelské rozhraní

Pro interakci uživatele s programem by mělo být vytvořeno vhodné uživatelské rozhraní. Nabízí se dvě možnosti. Buďto bude mít program grafické rozhraní nebo půjde ovládat přes příkazovou řádku. Nebo je možná kombinace obou předchozích možností a je jen na uživateli, které rozhraní si vybere. V případě příkazové řádky může být výhoda použití programu i na serverech či vzdálených počítačích bez grafického režimu. Pokud by nástroj používal grafické rozhraní, přidal by položku do systémové lišty Tray. Aplikace by tak byla jednoduše přístupná uživateli. Ikona aplikace v Tray liště by mohla zobrazovat aktuální stav. Pokud by chtěl uživatel provést pozastavení aplikace, použil by pro to kontextové menu, které by se zobrazilo při kliknutí na položku v Tray. Přes toto menu by uživatel mohl spravovat svazky, připojení k dalším uživatelům, řešit konflikty apod.

Kapitola 6

Implementace

Pro implementaci byly zvoleny jazyky C a C++. Pro C++ je použit standard C++0x. Jazyk C++ byl zvolen především kvůli svému zaměření, kterým je objektově orientované paradigma. Pro implementaci byly zvoleny knihovny, jejich přesné využití bude dále rozepsáno. Aplikace je navržena jako vícevláknová. Popis těchto vláken a synchronizace mezi nimi bude popsána dále.

6.1 Boost

Boost je soubor knihoven pro C++, které přináší abstraktnější použití a časté řešení problémů [1]. Některé části *Boost* byly přidány do TR1[1] a následně přidány do normy C++11. Z Boost knihoven byly pro tuto práci použity:

- *Boost.Asio*,
- *Boost.Filesystem*,
- Boost Smart pointers,
- *Boost.Thread*,
- *Boost.Uuid*.

Boost.Asio je knihovna, která přináší práci se sítí na abstraktnější úrovni. Knihovna je přenositelná i na další platformy [1]. Umožňuje škálovatelnost a jednoduché použití systémových prvků pro práci se sítí. Knihovna obsahuje také třídy pro použití SSL k zabezpečení komunikace. V knihovně je pro SSL použita knihovna *OpenSSL*. Všechny vlastnosti, které *OpenSSL* má, jsou přístupné i v této knihovně. Třída pro práci s SSL streamem sama neimplementuje žádné synchronizační mechanizmy [1]. V případě asynchronního použití musí být synchronizace vyřešena explicitně. V případě použití SSL streamu pouze v jednom vlákně se synchronizace nevyžaduje. Základem knihovny je třída `io_service`, která zajišťuje výměnu vstupně/výstupních operací mezi programem a samotným operačním systémem.

Boost.Filesystem zjednodušuje přístup k souborovému systému a je opět přenositelná na ostatní podporované platformy. Lze použít například rekurzivní čtení adresáře. Čtení adresářů používá koncept iterátorů. Lze jednoduše zjišťovat informace o velikosti a právech

[1] C++ Technical Report 1 (ISO/IEC TR 19768) popisuje další budoucí rozšíření standardní C++ knihovny.

k souboru nebo adresáři. Nevýhodou zůstává, že knihovna neimplementuje žádné další zjišťovací prostředky, které nejsou přenositelné na jiné platformy. Například neumí jednoduše zpřístupnit informace jako i-uzel, čas poslední modifikace či čas poslední změny i-uzlu, jelikož tyto atributy se vyskytují pouze na Unixu podobných systémech.

Boost Smart pointers umožňuje vytváření tzv. chytrých ukazatelů [1]. Tento chytrý ukazatel je nyní dostupný i v normě C++11. To nám do určité míry nahrazuje koncept garbage collectoru v jiných jazycích. Chytrý ukazatel je objekt, který si uchová námi vytvořený ukazatel, který používá dynamicky alokovanou paměť. Tento objekt také obsahuje čítač referencí. Pokud čítač referencí dosáhne 0, dynamicky alokovaná paměť bude automaticky uvolněna. Objekty typu `shared_ptr` je bezpečné používat ve vícevláknových aplikacích. Čtení může naráz probíhat z více vláken. Zápis může být proveden pouze určitými operátory. Jakýkoliv jiný simultánní přístup může vést k neočekávanému chování.

S chytrými ukazateli souvisí další třída `enable_shared_from_this`. Pokud náš objekt, který chceme dynamicky alokovat a použít v chytrém ukazateli, dědí od této třídy, při použití metody `shared_from_this` v našem objektu vytvoří automaticky chytrý ukazatel pro náš objekt. Metodu `shared_from_this` nelze použít v jediném případě, a to v konstruktoru daného objektu. Třída `enable_shared_from_this` je také součástí standardu C++11.

Boost.Thread umožňuje práci se samotnými vlákny a synchronizačními mechanizmy. Pro tuto práci byly použity třídy jako `mutex` a `condition_variable`. Třída `mutex` zajišťuje ochranu proti race condition a umožňuje tak používat výměnu dat mezi více vlákny. Třída `condition_variable` umožňuje mechanizmus, kdy jedno vlákno bude čekat, dokud mu nedojde oznámení od jiného vlákna.

Boost.Uuid umožňuje použití mechanizmu Universally unique identifier (UUID). Unikátnost takovéhoto vygenerovaného UUID je zaručena v případě, že se pro další vygenerované UUID nepoužilo jiného mechanizmu. Knihovna umožňuje generování těchto UUID a další převodní funkce pro jednoduché používání. [1]

6.2 Další knihovny

Dále byly pro implementaci synchronizačního nástroje použity tyto knihovny:

- SQLite,
- Protocol Buffers lite,
- *inotify-tools*,
- *OpenSSL*,
- *librsync*.

Práci s SQLite databází zajišťuje samotná C knihovna. Knihovna nepotřebuje pro své fungování žádný další server, jelikož pracuje přímo s databázovým souborem. V této práci bylo pro komunikaci použito jednoduchých C++ tříd projektu *sqdbcpp*[2], které zapouzdřují nativní funkce knihovny SQLite.

Jelikož tyto třídy neobsahovaly všechny funkce pro práci s SQLite, některé metody a třídy byly v mé práci doplněny. Doplněny byly především o přímý přístup k binárním polím tabulek a o metody pro převod do dalších základních typů. Databáze podporuje také

[2]Projekt je dostupný na adrese https://code.google.com/p/sqdbcpp/.

vícevláknový přístup. Mechanizmus zamykaní databáze musel být taktéž doplněn o jednoduché metody.

Další použitou knihovnou je *inotify-tools*. Tato C knihovna umožňuje jednodušší použití mechanizmu *inotify*. Umožňuje vytvoření sledování pro celý adresář rekurzivně, zjednodušuje práci pro přístup k samotným sledovaným položkám.

Knihovna Protocol Buffers byla již popsána v sekci 4.6.2. Knihovna *librsync* bude použita pro efektivní přenos souborů.

Knihovna *OpenSSL* byla použita pro generování X.509 certifikátů a klíčů. Samotná knihovna zpřístupňuje celou řadu funkcí pro kryptografii. Knihovnu používá také knihovna *Boost.Asio*.

6.3 Architektura aplikace

Pro zjednodušení implementace programu bylo rozhodnuto použít rozhraní příkazové řádky. Aplikace je navržena jako vícevláknová, jelikož je potřeba reagovat asynchronně na síťové události a události v souborovém systému. Dále je třeba použít blokující operace pro uživatelský vstup. Celkově tak jsou potřeba minimálně tři vlákna pro tyto nezávislé operace a hlavní vlákno.

Hlavní vlákno zajišťuje správu svazků a mělo by obsahovat logiku pro řízení celé aplikace. Hlavní smyčka v hlavním vlákně také zajišťuje sbírání požadavků třídy `FSMonitor` a po větších shlucích jsou přidávány přes třídu `VolumeController` do jednotlivých svazků. Tím je zaručeno, že pokud bude existovat více změn najednou, budou všechny přidány pouze s jedním snímkem. Pokud by totiž byly přidávány změny postupně po jedné, pro každou takovou změnu by bylo zapotřebí vytvořit nový snímek.

Po spuštění programu dojde k inicializaci uživatele, pokud nebyl ještě vytvořen, a vygeneruje se mu certifikát s privátním klíčem. Dále dojde ke kontrole uživatelských svazků, pokud uživatel nějaké vlastní. Dále se vytvoří postupně tři vedlejší vlákna. Jedno vlákno je pro síťovou komunikaci a vyřizování vzdálených požadavků, další vlákno je pro uživatelský vstup a poslední vlákno čeká na události ze souborového systému. Pro synchronizaci mezi vlákny byly použity mutexy a v některých případech sdílené fronty.

6.3.1 Rozvržení tříd

Pro návrh programu bylo použito objektově orientovaného paradigma. Všechny třídy, které byly v programu použity, budou dále podrobněji popsány.

Servent

Třída `Servent` symbolizuje jeden z uzlů v topologii peer–to–peer. Třída zajišťuje vytvoření síťového připojení směrem k jiným uzlům a zároveň naslouchá na zvoleném portu na příchozí připojení. Naslouchání pro příchozí připojení zde zajišťuje třída `acceptor` z knihovny *Boost.Asio*. Ten je vytvořen z třídy `io_service`. Při vytváření třídy `Servent` se načte uživatelův certifikát a primární klíč a vytvoří se SSL kontext. Při vytváření kontextu dojde k zakázání zavržených protokolů SSLv2 a SSLv3. Pro zabezpečení komunikace může být použit některý z verzí TLS. Po vytvoření instance třídy `Servent` třída zajišťuje veškerou komunikaci. Pokud je zapotřebí vytvoření odchozího připojení, použije se třída `Session`, která je popsána dále. Stejně tak jedna instance třídy `Session` může představovat příchozí připojení. Třída `Servent` si ukládá všechny tyto instance do vnitřní fronty. Všechny metody

třídy `Servent`, které pracují s instancí třídy `io_service`, používají pro synchronizaci `mutex`.

Session

Třída `Session` obstarává asynchronní komunikaci po vytvořeném socketu. Pro vyloučení vícenásobného přístupu k socketu byl použit `mutex`. Třída obsahuje vnitřní frontu pro odesílání zpráv. Zprávu zde představuje třída `Message`. Třída `Session` obsahuje také instanci typu `Protocol`, která řídí komunikaci pro dané připojení. Čtení zpráv probíhá následovně. Nejprve budou přečteny 4 bajty ze streamu, které představují velikost zprávy. Dále se bude číst tolik bajtů, kolik bylo určeno v délce zprávy. Horní hranice délky zprávy byla po dohodě s ostatními studenty v rámci kompatibility určena na 327680 bajtů. Po přečtení zprávy se oznámí třídě `Protocol` příchod nové zprávy. Třída `Session` zajišťuje také autentizaci kontrolou otisku certifikátu pomocí metod třídy `PeerController`. Pokud nebude při příchozím nebo odchozím připojení otisk certifikátu nalezen v lokální databázi, bude o tom informovat uživatele. Je jen na uživateli, zda otisk přijme. Po přijetí otisku se otisk uloží do lokální databáze pro příští použití. Pro synchronizaci uživatelského vstupu a čekání na potvrzení otisku je zde použito `condition_variable` spolu s třídou `mutex`.

Message

Třída `Message` představuje zprávu v komunikaci. Tato třída zapouzdřuje zprávu protokolu Sync, který byl vygenerován z předpisu protokolu v syntaxi Protocol Buffers (PB). Zpráva PB je podle potřeby serializována do vyrovnávací paměti nebo parsována z přijatých dat.

Protocol

Tato třída na základě příchozí zprávy generuje okamžitě odpověď a tu předá zpět třídě `Session`. Při příchodu nové zprávy je nejprve zkontrolován obsah zprávy podle jeho deklarovaného typu. Pokud tyto údaje nesouhlasí, zpráva bude ignorována. Třída `Protocol` si uchovává veškerá data, která získala v průběhu komunikace. Dále obsahuje čítač transakcí a mapu transakcí, kde uchovává pár UUID svazku spolu s UUID dotazovaného objektu v daném svazku.

SafeQueue

Generická třída `SafeQueue` představuje sdílenou frontu. Toto třída byla vytvořena za účelem jednoduchého synchronizačního mechanizmu mezi vlákny. Třída obsahuje instance tříd `mutex` a `condition_variable`. Dále obsahuje samotnou frontu. V případě vybírání položek z fronty se nejdříve zkontroluje, zda není vnitřní fronta prázdná, aby nedošlo k zablokování.

Peer

Třída `Peer` shlukuje informace o uzlu a zajišťuje metody pro práci s certifikáty. Dále zařizuje vytváření položek entit *peer* a *connection* do databáze.

PeerController

Tato třída představuje manažera všech uzlů. Jedna instance třídy `Peer` je rezervována pro lokálního uživatele. Dále je uložen seznam všech ostatních uživatelů. Třída zajišťuje metody pro hledání ostatních uživatelů a metody pro ověření otisků uživatelů.

FSEvent

Struktura `FSEvent` představuje konkrétní událost souborového systému. Struktury jsou inicializovány v třídě `FSMonitor`. Struktura identifikuje entitu v souborovém systému pomocí absolutní cesty. Dále obsahuje typ události a podrobnosti o entitě. V případě události pro přejmenování je uložena i původní cesta k entitě.

FSMonitor

Třída `FSMonitor` zajišťuje generování událostí ze souborového systému. Zapouzdřuje nižší funkce pro inicializace a destrukci sledovaných entit v souborovém systému. V metodě run je smyčka, která čeká na příchozí události. Z těch jsou potom vytvořeny konkretní události `FSEvent`. Pokud do jedné sekundy nepřijde nová událost, smyčka pokračuje, aby se nezablokovala. `FSMonitor` reaguje ještě na příchozí požadavky třídy `VolumeController` přes sdílenou frontu při vytvoření a smazání svazku. Podle toho, jaká akce byla se svazkem provedena, bude monitor vytvářet sledování souborového systému nebo rušit již vytvořené sledování.

VolumeController

Třída `VolumeController` pracuje s třídami `Volume`. Po spuštění dojde k nalezení uživatelových svazků. Ty jsou po celou dobu spuštění programu udržovány v interním seznamu. `VolumeController` vytváří a ruší zadané svazky. Pokud je svazek vytvořen nebo smazán, oznámí třídě `FSMonitor` přes sdílenou frontu danou změnu. Všechny příchozí události třídy `FSMonitor` jsou do svazků přidávány přes tuto třídu. `VolumeController` totiž musí podle cesty v souborovém systému dané entity najít příslušný svazek.

Volume

Třída `Volume` představuje jednotlivý svazek. Obstarává vytváření a mazaní jednotlivých svazků z databáze. Dále provádí kontrolu položek svazku. Existují dva typy kontrol. Jedna úplná kontrola probíhá vždy při vytvoření svazku nebo při spuštění nástroje. Další částečná kontrola zpracovává události souborového systému, které se nashromáždily pro jednotlivý svazek. Oba typy kontrol provádí porovnání metadat skutečných položek souborového systému s lokální databází. Pokud bude detekována změna, bude vytvořen nový snímek spolu se změněnými položkami.

Snapshot

Třída `Snapshot` představuje jednotlivý snímek. Třída obstarává vytváření snímků v databázi. V případě vytvoření nového snímku si třída udržuje seznam instancí tříd `Metadata`. Nejprve je vytvořen snímek a následně jsou vytvořeny položky metadat. Vytvoření snímku s metadaty je uzavřeno v transakci.

Metadata

Tato třída představuje samotná metadata. Třída provádí komparaci metadat s údaji z databáze, které představuje. Třída obstarává také vytvoření nových položek v databázi.

Data

Třída `Data` pracuje se samotnými binárními daty v databázi. Jelikož mají data vlastní identifikátor, jsou modelována jako další entita.

6.4 Popis implementace a testování

Implementaci samotné aplikace předcházelo vytvoření několika prototypů na otestování síťového připojení a vzájemné komunikace. Dále byl vytvořen jednoduchý prototyp pro synchronizaci souborů. Implementace aplikace probíhala v několika etapách a v každé z těchto etap byla otestována funkčnost daného podsystému. Všechny etapy postupného vývoje budou nyní popsány.

Pro testování aplikací byla použita distribuce Gentoo Linux[3]. Implementace jednotlivých prototypů a samotné aplikace probíhala v programu Qt Creator IDE[4]. Tento program nabízí také debugger pro ladění aplikací, který byl používán pro ověření funkčnosti programu. Pro ladění a ověření funkčnosti aplikace bylo použito programu *valgrind*[5].

6.4.1 Prototyp - síťová komunikace a jednoduché zprávy

První prototyp představoval implementaci síťového modelu, který byl použit i v dalším prototypu a samotné aplikaci. Cílem bylo otestovat síťovou komunikaci s ostatními klienty za pomocí jednoduchých zpráv, definovaných v ASN.1. Knihovna *libtasn1*, která byla pro implementaci prototypu ze začátku použita, se ukázala jako problém. Implementace vygenerování a parsování jednoduché zprávy za pomocí této knihovny byla náročnější, než se původně očekávalo. S ostatními studenty implementujícími podobný nástroj, bylo nakonec domluveno, že pro zapouzdření zpráv je nutné použít jiné technologie, jelikož na ostatních platformách byly podobné problémy s použitím ASN.1.

Pro zapouzdření jednoduchých textových zpráv byla nakonec použita knihovna Protocol Buffers. Jednoduchá struktura Protocol Buffers byla zapouzdřena do jednoduché lenght-value struktury, která byla použita i v samotné konečné aplikaci. Posílání jednoduchých textových zpráv bylo otestováno na dvou instancích prototypu na lokálním počítači. Později byl prototyp také otestován při komunikaci přes Internet s podobným nástrojem na platformě Android. Tímto byla úspěšně otestována síťová architektura programu, která byla použita v dalším prototypu i finální verzi aplikace.

6.4.2 Prototyp - jednoduchá synchronizace souborů

Jako další prototyp se jevila jednoduchá aplikace pro synchronizaci souborů. Nejprve byl doplněn původní protokol prvního prototypu o struktury pro synchronizaci souborů. Použita byla opět knihovna Protocol Buffers. Samotné struktury vytvořené za pomocí knihovny Protocol Buffers byly zapouzdřeny do TLV struktury. Tím měla každá zpráva jasně definovanou délku a typ.

Detekce změn probíhala pomocí atributů souborů jako modifikační čas, velikost a název souboru. Adresáře byly ignorovány. Synchronizace probíhala tímto způsobem. První prototyp odeslal seznam změn druhému prototypu. Druhý prototyp zkontroloval svůj pracovní adresář. V případě, že atributy některých položek nesouhlasily s atributy v přijatém

[3] Distribuce je dostupná na adrese `http://www.gentoo.org`.
[4] Nástroj je dostupný na adrese `http://qt.digia.com/Product/Developer-Tools/`.
[5] Nástroj je dostupný na adrese `http://valgrind.org/`.

seznamu, druhý prototyp ihned odeslal požadavek na změněný soubor. První prototyp na tuto zprávu reagoval odesláním samotných dat souboru. Druhý prototyp takto pokračoval, dokud nesynchronizoval celý seznam.

6.4.3 Finální aplikace

Implementace samotné aplikace probíhala v několika etapách, kde po dokončení určité etapy bylo otestováno, zda jednotlivý subsystém pracuje správně.

Cílem první etapy bylo implementovat databázi metadat a rozhraní pro přístup k těmto metadatům i samotným datům. Dále následovala implementace hromadného zálohování po spuštění aplikace. Pro testování byl v operačním systému vytvořen adresář s několika vytvořenými soubory. V nástroji byl vytvořen svazek s cestou k tomuto adresáři. Při spuštění programu došlo k porovnání svazku s lokální databází a případnému přidání či odebrání položek z databáze. Položky z databáze se nikdy nemazaly, odebírání má zde význam pouze jako atribut metadat. Testování dále probíhalo na různě vytvořených adresářích a položkách. Po každém testu bylo zkontrolováno, zda se v databázi vytvořila očekávaná metadata.

Další etapou byla implementace zálohování souborů v reálném čase. Testování probíhalo na spuštěné aplikaci. Opět byl vytvořen svazek s reálným pracovním adresářem. Funkčnost byla testována následovně. Očekávané chování aplikace bylo, že v případě provedení změn v pracovním adresáři se tyto změny zachytí a následně zpracují porovnáním a přidáním do databáze. Při tomto testu byly prováděny změny na položkách v pracovním adresáři a bylo sledováno, zda aplikace provádí očekávanou reakci. Testy pro základní operace s položkami jako je přidávání, změna a mazaní souboru dopadly podle očekávání.

Cílem další etapy byla implementace jednoduchého uživatelského rozhraní, správy uživatelů a certifikátů. Nejprve byly vytvořeny metody pro generování uživatelských certifikátů. Vygenerovaný certifikát se následně použije pro uživatele. Generování probíhá při prvním spuštění programu, kdy také uživatel zadá vlastní název. Testování vygenerovaného certifikátu proběhlo pomocí programu *openssl*. Pomocí tohoto programu bylo zkontrolováno, zda všechny údaje v certifikátu odpovídají hodnotám, které byly zadány při generování.

Další etapa představovala samotnou synchronizaci dat mezi uzly. Nejprve bylo třeba v aplikaci implementovat protokol pro výměnu dat. Testování výměny dat probíhalo na jediném počítači, kde byly spuštěny dvě instance synchronizačního nástroje. Jeden z nástrojů byl ručně připojen k druhému. Oba nástroje měly vytvořeny svazky se stejným identifikátorem, ale jiným pracovním adresářem v souborovém systému. Po zadání příkazu *pull* se začalo synchronizovat. Po dokončení synchronizace se systémovým příkazem *diff*[6] rekurzivně porovnaly oba pracovní adresáře, zda jsou totožné.

V další etapě byla implementována obnova souborů. Pro obnovu souboru je nutné nejdříve nalézt požadovanou verzi souboru a jeho identifikátor. Obnovu lze provést po zadání identifikátoru souboru a cesty v souborovém systému. Testování podoby obnoveného souboru probíhalo pomocí příkazu *diff*.

Cílem další etapy bylo implementovat efektivní přenos dat mezi uzly. Toho bylo dosaženo knihovnou *librsync*. Nejprve bylo nutné implementovat rozhraní pro práci s knihovnou. Testování implementace probíhalo následujícím způsobem. Do pracovního adresáře se přidal náhodný soubor, který se zálohoval. Dále se k tomuto souboru přidala náhodná data a soubor se opět zálohoval. Pak se k tomuto uzlu připojil další uzel. Druhý uzel si vynutil synchronizaci a bylo sledováno, zda se při stahování druhé verze souboru použilo delta algoritmu. V případě, že se algoritmu použilo, stahovaná data pro druhý snímek by měla

[6] Nástroj a manuál je dostupný na adrese https://www.gnu.org/software/diffutils/.

být ta, která se náhodně přidala k tomuto souboru u prvního uzlu. Oba pracovní adresáře byly po synchronizaci porovnány pomocí příkazu *diff*, čímž se otestovalo zpětné sestavení souboru.

Poslední etapa zahrnovala implementaci automatické synchronizace. Nástroj v případě detekce změny shromáždí přidaná metadata a následně je pošle všem právě připojeným uzlům. Ostatní uzly zkontrolují, zda tato metadata již v databázi nemají. V případě, že tato metadata nevlastní, dotážou se na ně zpět uzlu, který je poslal. Po přijetí kompletních metadat si nástroj zavede tato metadata pod snímkem, který má stejný identifikátor jako svazek, ke kterému náleží. Tento anonymní snímek má stejný identifikátor jako svazek, ke kterému náleží. Tím lze jednoduše dohledat anonymní snímky. Takto se ukládají všechna metadata, pokud jsou uzly propojeny. Synchronizace snímků proběhne až při příštím připojení, kdy se nástroj automaticky dotáže na snímky, které nevlastní. Synchronizaci snímků lze také vyvolat ručně přes příkazové rozhraní. Testování změn probíhalo jako v předchozích případech.

6.5 Testování

Nyní bude popsán postup pro testování aplikace. Pro testování byl vytvořen náhodný vzorek dat pomocí systémového příkazu *dd*. Ukázka vytvoření náhodného souboru o velikosti 1MB dat: `dd if=/dev/urandom of=data bs=1 count=1000000`. Tímto způsobem byla vytvořena sada souborů 1MB, 10 souborů po 1MB, 100MB a 5 souborů po 100MB.

Testování proběhlo postupně pro tři různé případy. V prvním případě bylo testováno zálohování souborů na jediném uzlu. Dále byla testována synchronizace dat mezi dvěma uzly na jednom počítači a potom také mezi dvěma počítači na lokální bezdrátové síti. Při synchronizaci dat byly testovány oba režimy přenosu souborů, a to úplný přenos souboru a přenos souboru za pomoci delta algoritmu rsync.

Pro testování bylo použito na všech počítačích prostředí operačního systému Gentoo Linux. Při testování se zjišťovala doba, za jakou je nástroj schopen provést požadovanou činnost. Pro srovnání jsou také uvedeny hodnoty dosažené pomocí jiného programu. Testování proběhlo pro každou sadu náhodných souborů. Pro testování bylo do nástroje implementováno zaznamenání času při začátku požadované akce a po provedení požadované činnosti se vypočetl absolutní čas, který byl potřebný pro tuto akci.

6.5.1 Zálohování

Pro testování zálohování byl nástroj postupně spuštěn pro každou sadu náhodných dat. Nejprve se inicializoval program, kdy se vygeneroval potřebný certifikát. Poté byl vytvořen svazek z jedné požadované sady náhodných souborů. Postupně takto vznikly čtyři svazky s pracovním adresářem, který obsahoval náhodnou sadu dat. Při vytváření svazku bylo sledováno, za jakou dobu bude pracovní adresář zálohován. Po dokončení zálohování byl program vždy ukončen a spuštěn znovu a zaznamenala se doba, která byla nutná pro opětovnou kontrolu pracovního adresáře pomocí metadat. Pro srovnání bylo vždy na stejné sadě souborů použito systémového příkazu *cp*. Výsledky testování jsou uvedeny v tabulce 6.1.

Z výsledků testů v tabulce 6.1 je patrné, že činnost zálohování oproti klasickému kopírování souborů v souborovém systému je pomalejší. To může být způsobeno kontrolou metadat a režií zápisu do databáze. Při opětovné kontrole metadat je čas již zanedbatelný.

velikost dat [MB]	Implementovaný nástroj			Příkaz *cp*	
	doba zálohování [s]	opětovná kontrola [s]	průměrná rych. [MB/s]	doba kopírování [s]	průměrná rych. [MB/s]
1	0,290	0,003	3,448	0,098	10,204
10 x 1	0,959	0,004	10,428	0,417	23,981
100	5,104	0,003	19,592	2,256	44,326
5 x 100	24,957	0,004	20,035	20,430	24,474

Tabulka 6.1: Tabulka výsledků testování zálohování

6.5.2 Synchronizace na lokálním počítači

Druhý test se zabývá testováním synchronizace dat mezi dvěma uzly na jediném počítači. Pro test byly použity již vytvořené instance nástroje při zálohování. Použitá sada dat je vždy stejná jako u prvního testu. Pro testování byla vždy pro každou sadu dat vytvořena další instance nástroje. V průběhu testu byly však maximálně spuštěny dvě instance nástroje. Při tomto testu bylo vždy sledováno, jaká doba je potřebná pro zjištění změn, které nástroj nemá, stažení samotných dat a následné vytvoření položek v souborovém systému. Všechny tyto činnosti jsou zahrnuty v době synchronizace.

Nejprve byla provedena synchronizace pro samotné sady dat a zaznamenaly se údaje, kdy nástroj musí kopírovat vždy všechna data. Po každé synchronizaci byla synchronizace spuštěna znovu, aby se zjistilo, jaká doba je potřebná pro opětovnou detekci změn.

Další částí tohoto testu bylo otestování použití delta algoritmu rsync. Nejprve byly všechny instance nástrojů se zálohami zduplikovány pro pozdější test. Potom bylo u každé sady původních náhodných dat změněno 10% původního obsahu každého testovaného souboru. Těchto 10% bylo vždy změněno od poloviny souboru. Pro změnu dat bylo použito systémového příkazu *dd*. Ukázka příkazu pro změnu souboru o velikost 1MB, kdy od poloviny souboru bude zapsáno 100kB náhodných dat:
`dd if=/dev/urandom of=data seek=500000 conv=notrunc bs=1 count=100000`.

Po modifikaci dat byl vždy spuštěn nástroj pro zvolenou sadu náhodných dat. Data se opět zálohovala. Na druhém uzlu se spustila instance pro sadu dat z předchozí části tohoto testu. Na jedné straně byl tedy uzel, který měl již modifikovaná původní data a na druhé straně byl uzel, který měl pouze původní data. Jelikož oba uzly vlastní stejná původní data, bude použito delta algoritmu rsync. Druhý uzel pošle požadavek na data společně se signaturou svých dat. První uzel na základě znalosti signatury těchto dat pošle pouze změny v upraveném souboru oproti původnímu souboru. Druhá strana na základě přijatých změn a vlastního původního souboru zkonstruuje nový soubor. Při každé takové synchronizaci byla zaznamenána doba synchronizace.

Výsledky obou částí testů jsou uvedeny v tabulce 6.2. Z výsledků testu je vidět jasný rozdíl mezi synchronizací, kdy se stahují všechna data a synchronizací, kdy se stahují pouze změny dat. Doba pro zjištění opětovných změn je již zanedbatelná.

6.5.3 Synchronizace mezi dvěma počítači v lokální síti

Třetí test je obdobou předchozího testu s tím rozdílem, že probíhal již na dvou různých počítačích. Na obou těchto počítačích bylo pro testování použito obdobné prostředí operačního systému jako v předchozích testech. Počítače byly připojeny v lokální bezdrátové síti. Schéma zapojení sítě je na obrázku 6.1.

velikost dat [MB]	Implementovaný nástroj					
	Synchronizace dat			**Synchro. dat s delta alg.**		
	doba sync. [s]	opětovná sync. [s]	průměrná rych. [MB/s]	doba sync. [s]	opětovná sync. [s]	průměrná rych. [MB/s]
1	1,794	0,120	0,557	1,207	0,161	0,829
10 x 1	14,476	0,081	0,691	9,290	0,162	1,076
100	97,982	0,081	1,021	24,402	0,161	4,098
5 x 100	470,667	0,082	1,062	146,493	0,208	3,413

Tabulka 6.2: Tabulka výsledků testování synchronizace dat za použití jednoho počítače

Obrázek 6.1: Zapojení lokální bezdrátové sítě

Testování mělo stejný průběh jako při předchozím testu popsaném v kapitole 6.5.2. Pro srovnání síťového přenosu bylo také zjišťováno, za jakou dobu budou přenesena data pomocí programu *scp*. Pro vzájemnou autentizaci bylo použito klíče. Udané časy pro příkaz *scp* zahrnují absolutní čas, který byl potřebný pro běh tohoto programu vždy pro určitou sadu souborů. Všechny výsledky jsou uvedeny v tabulce 6.3.

Z výsledků v tabulce 6.3 je patrné mírné zhoršení oproti testu na jednom počítači. Synchronizační nástroj v porovnání s příkazem *scp* vykazuje při přenosu objemnějších dat větší režii. V případě použití delta algoritmu rsync se však objem přenášených dat eliminuje, a tím se ušetří přenosové pásmo. Tento algoritmus lze však použít jen za podmínky, kdy obě strany vlastní původní soubor.

velikost dat [MB]	Implementovaný nástroj						**Příkaz *scp***
	Synchronizace dat			**Synchro. dat s delta alg.**			
	doba sync. [s]	opět. sync. [s]	prům. r. [MB/s]	doba sync. [s]	opět. sync. [s]	prům. r. [MB/s]	doba kop. [s]
1	2,083	0,110	0,480	1,207	1,043	0,829	1,008
10 x 1	15,027	0,101	0,665	6,765	1,561	1,478	9,178
100	141,471	0,102	0,707	47,648	0,256	2,099	83,902
5 x 100	671,387	0,105	0,744	148,563	0,197	3,366	403,416

Tabulka 6.3: Tabulka výsledků testování synchronizace dat mezi dvěma počítači v lokální síti

Kapitola 7

Závěr

Cílem této práce bylo navržení a implementování synchronizačního a zálohovacího nástroje. Toho bylo dosaženo po nastudování existujících obdobných nástrojů a metod, které tyto nástroje používají. Nezbytnou částí pro návrh nástroje bylo také nastudovaní možností, které se nabízí na operačním systému Linux.

Aplikace byla otestována na náhodně vytvořeném vzorku dat. Tím se otestovala její základní funkčnost zálohování a synchronizace dat. Dosažené výsledky byly porovnány s výsledky dosaženými základními programy pro kopírování dat v lokálním souborovém systému a po lokální síti. Zlepšení efektivity přenášených dat bylo dosaženo pomocí delta algoritmu rsync. Při použití delta algoritmu rsync se při výměně dat posílají jen skutečně změněná data, což šetří čas a přenosové pásmo. Z výsledků je však znatelné, že použitá implementace a řešení přenosu souborů má větší režii než klasické kopírování souborů. Tato režie je způsobena přístupy do databáze a dále neefektivní implementací aplikace.

Zpracování souborů bylo neefektivně implementováno a samotná aplikace spotřebovává velké množství operační paměti. Tento problém nastává hlavně při práci s objemnými daty v řádech stovek MB. Všechny soubory jsou před posláním a po přijetí uloženy v operační paměti. Tento nedostatek by bylo možné odstranit použitím dočasných souborů a čtením souborů v případě potřeby. To by však potřebovalo do aplikace implementovat řízení komunikačního protokolu a zpracování transakčních zpráv nezávisle na přijetí těchto zpráv. Z těchto důvodu aplikace ukládá soubory maximálně do objemu 1GB dat a ukládání pro větší soubory nebylo implementováno. Komunikační protokol umí nad rámec zadání práce řešit konfliktní soubory, to však z důvodů velké složitosti nebylo v nástroji na této platformě implementováno. Pro lepší řešení ukládání dat by bylo také vhodné použít databázi s podporou rekurzivních dotazů pro jednodušší hledání v hierarchii metadat.

Budoucí rozšíření programu by se mělo zabývat zlepšením jejich současných nedostatků, které byly popsány. Aplikace by mohla být rozšířena o grafické rozhraní. Dalším rozšířením aplikace by mohlo být překonání NATu v síti za pomoci třetího uzlu. Zde by však bylo nutné rozšíření současného protokolu. Dále by bylo užitečné, kdyby aplikace měla možnost nalézt další kompatibilní aplikace spuštěné ve stejné lokální síti.

Literatura

[1] Boost C++ Libraries. [online], 2007, [cit. 2013-05-03].
URL http://www.boost.org/

[2] SQLite Home Page. [online], 2013, [cit. 2013-05-03].
URL https://www.sqlite.org/

[3] Barrett, D.; Silverman, R.; Byrnes, R.: *SSH, The Secure Shell: The Definitive Guide*. Definitive Guide Series, O'Reilly Media, Incorporated, 2005, ISBN 9780596008956.

[4] Breda, L.: Grive [Grive]. [online], 2013, [cit. 2013-05-03].
URL http://www.lbreda.com/grive/start

[5] Chacon, S.: *Pro Git*. CZ.NIC, z. s. p. o., 2009, ISBN 978-80-904248-1-4.

[6] Cohen, M.: *Take Control of Syncing Data in Snow Leopard, 1st Edition*. Take control, TidBITS Publishing, Incorporated, 2009, ISBN 9781615420032.

[7] Cougias, D.; Heiberger, E.; Koop, K.: *The Backup Book: Disaster Recovery from Desktop to Data Center*. Network Frontiers Field Manual Series, Schaser-Vartan Books, 2003, ISBN 9780972903905.

[8] Dierks, T.; Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Internet Engineering Task Force, 2008.

[9] Dubuisson, O.: *ASN.1: Communication Between Heterogeneous Systems*. 2000, ISBN 0-12-6333361-0.

[10] Ford, B.; Srisuresh, P.; Kegel, D.: Peer-to-Peer Communication Across Network Address Translators. Technická zpráva, 2005, [cit. 2013-05-03].
URL http://www.brynosaurus.com/pub/net/p2pnat/

[11] Georgiev, M.; Iyengar, S.; Jana, S.; aj.: The most dangerous code in the world: validating SSL certificates in non-browser software. In *ACM Conference on Computer and Communications Security*, 2013, [cit. 2013-05-03].
URL
https://crypto.stanford.edu/~dabo/pubs/abstracts/ssl-client-bugs.html

[12] Google: Google Drive Help. [online], 2013, [cit. 2013-05-03].
URL https://support.google.com/drive/

[13] Google: Google Drive SDK-Google Developers. [online], 2013, [cit. 2013-05-03].
URL https://developers.google.com/drive/

[14] Google: Protocol Buffers - Google Developers. [online], 2013, [cit. 2013-05-03]. URL https://developers.google.com/protocol-buffers/

[15] Gutmann, P.: cryptlib Encryption toolkit. [online], 2013, [cit. 2013-05-03]. URL http://www.cs.auckland.ac.nz/~pgut001/cryptlib/

[16] Halsall, F.: *Computer Networking And the Internet*. China Machine Press, 2005, ISBN 9780321263582.

[17] Karmazín, J.: *Synchronization and Backup of Data under Android OS*. Bakalářská práce, FIT VUT v Brně, Brno, 2013.

[18] Kerrisk, M.: *The Linux Programming Interface: A Linux and Unix System Programming Handbook*. No Starch Press Series, No Starch Press, 2010, ISBN 9781593272203.

[19] Loeliger, J.; McCullough, M.: *Version Control with Git: Powerful tools and techniques for collaborative software development*. Version Control With Git, O'Reilly Media, Incorporated, 2012, ISBN 9781449316389.

[20] Love, R.: *Linux System Programming: Talking Directly to the Kernel and C Library*. Oreilly Series, O'Reilly Media, 2007, ISBN 9780596009588.

[21] McCabe, J.: *Network Analysis, Architecture, and Design*. The Morgan Kaufmann Series in Networking, Elsevier Science, 2010, ISBN 9780080548753.

[22] Nelson, S.: *Pro Data Backup and Recovery*. Apresspod Series, Apress, 2011, ISBN 9781430226628.

[23] Pilato, M.; Collins-Sussman, B.; Fitzpatrick, B.: *Version Control with Subversion*. Oreilly Series, O'Reilly Media, 2008, ISBN 9780596510336.

[24] Preston, C.: *Backup & Recovery: Inexpensive Backup Solutions for Open Systems*. Oreilly Series, O'Reilly Media, 2008, ISBN 9780596102463.

[25] Siever, E.; Figgins, S.; Love, R.; aj.: *Linux in a Nutshell*. LINUX IN A NUTSHELL, O'Reilly Media, Incorporated, 2009, ISBN 9780596154486.

[26] Slováček, J.: *Synchronizace a zálohování dat pod Windows*. Bakalářská práce, FIT VUT v Brně, Brno, 2013.

[27] Tridgell, A.: *Efficient Algorithms for Sorting and Synchronization*. Dizertační práce, The Australian National University, February 1999.

[28] Tridgell A., P. M.: The rsync algorithm. [online], 1998, [cit. 2013-05-03]. URL https://rsync.samba.org/tech_report/tech_report.html

[29] Wolf, C.: Cluster synchronization with Csync^2. [online], 2013, [cit. 2013-05-03]. URL http://oss.linbit.com/csync2/

Příloha A

Obsah CD

application/		Zdrojové kódy projektu Sync
	bin/	Adresář obsahuje spustitelnou aplikaci Sync
	build/	Adresář pro kompilaci
	include/	Adresář s hlavičkami
	src/	Adresář se zdrojovými kódy
	sql.txt	SQL skript pro vytvoření databáze
	Sync.db	Soubor s prázdnou databází
	Sync.proto	Soubor s popisem protokolu
latex/		Zdrojové soubory této práce
MANUAL_Sync.txt		Uživatelský manuál
xmarta02.pdf		Tato práce ve formátu PDF

Příloha B

Uživatelský manuál

V této kapitole bude popsáno základní užití nástroje *Sync*. Informace jsou dostupné také po zadaní příkazu *help* po spuštění aplikace. Tento uživatelský manuál je obsažen také na přiloženém CD, kde je rozšířen o pár příkladů.

B.1 Kompilace a instalace

Instalaci programu *Sync* lze provést následujícím způsobem. Před samotným procesem kompilace je zapotřebí mít v systému nainstalovány tyto knihovny:

- *boost* 1.49.0,
- *protobuf* 2.4.0a,
- *sqlite* 3.7.16.2,
- *openssl* 1.0.1c,
- *inotify-tools* 3.13,
- *librsync* 0.9.7.

U každé knihovny je uvedena verze, na které byl nástroj při vývoji testován. Pro kompilaci je potřeba mít v systému také program *cmake* a *gcc* minimálně ve verzi 4.6.3.

Po okopírování zdrojových kódů aplikace se přesuneme do adresáře s těmito kódy a kompilaci provedeme následovně.

```
cd build
cmake ..
make
```

Samotný nástroj bude po kompilaci vytvořen v adresáři `build/bin` pod názvem *Sync*.

B.2 Použití programu

Před prvním spuštěním programu *Sync* je nutné nejdříve vytvořit databázi. Prázdná databáze je již připravena v adresáři se zdrojovými kódy. Případně lze databázi vytvořit ručně z přiloženého SQL skriptu *sql.txt*, který je také přiložen v adresáři se zdrojovými kódy.

Program *Sync* před spuštěním nástroje musí mít databázový soubor *Sync.db* v pracovním adresáři nástroje.

Samotný nástroj lze spustit s těmito parametry:
`Usage: Sync [-t] port.`
Povinný parametr je port, na kterém bude aplikace naslouchat příchozím požadavkům. Program lze spustit s dobrovolným parametrem *-t*, který bude zobrazovat čas, který byl nutný pro vytvoření transakcí při zálohování a synchronizaci.

Po prvním spuštění nástroje *Sync* je nutné zadat pracovní název relace. Dále lze program ovládat pomocí příkazů v konzoli:

- *connect(c) ipv4,ipv6 port* - po zadání IPv4 nebo IPv6 adresy a portu se pokusí připojit ke vzdálenému uzlu;
- *disconnect(d) session_index* - odpojí se od uzlu, index lze zjistit ve výpisu připojení;
- *Yes session_index* - autentizuje uzel s daným indexem připojení;
- *no session_index* - odmítne připojení s daným indexem;
- *myself(m)* - vypíše informace o vlastním uzlu;
- *list(l) sessions(s)* - vypíše informace o aktuálních připojeních a jejich indexy;
- *list(l) volumes(v)* - vypíše informace o lokálních i vzdálených svazcích a jejich indexy;
- *list(l) peers(p)* - vypíše informace o uzlech;
- *list(l) versions(i) volume_index item_with_path* - vypíše seznam verzí souborů pro daný svazek s daným názvem a cestou ve svazku;
- *restore(r) uuid restore_absolute_path* - obnoví soubor z databáze do zadané absolutní cesty v souborovém systému, je nutné nejdříve vyhledat požadovaný identifikátor souboru;
- *volume(v) name absolute_path* - vytvoří svazek z existujícího adresáře v souborovém systému, je nutné zadat vlastní název svazku a absolutní cestou k adresáři v souborovém systému;
- *updates(u) volume_index* - zapne nebo vypne automatické aktualizace svazku, příznak lze zjistit ve výpisu svazků;
- *pull* - bude vynucena synchronizace s ostatními připojenými uzly a jejich stejnými svazky;
- *volumefrom(vf) name absolute_path session_index volume_index* - vytvoří svazek z existujícího připojení;
- *rvolume name* - odstraní všechna data v databázi k zadanému svazku spolu se samotným svazkem, pracovní adresář nechá beze změny;
- *verbose* - zapne nebo vypne podrobné výpisy;
- *vverbose* - zapne nebo vypne vypisování dotazů pro databázi;
- *help(h)* - vypíše nápovědu.

U některých příkazů je v závorce uvedena zkratka pro daný příkaz.

Příloha C

Protokol

Zde je uvedena kompletní definice komunikačního protokolu v syntaxi Protocol Buffers. Protokol byl vytvořen ve spolupráci s ostatními studenty z fakulty v rámci bakalářských prací s podobnou tématikou [17] [26].

```
package Sync;

option optimize_for = LITE_RUNTIME;

// typy zpráv
enum MessageType {
    ERROR          =  2;
    HELLO          =  3;
    ADDED_VOLUME   =  4;
    REMOVED_VOLUME =  5;
    UPDATE         =  6;
    SYNCHRONIZED   =  7;
    SUCCESS        =  8;
    PULLED         =  9;
    SNAPSHOT       = 10;
    METADATA       = 11;
    DATA           = 12;
    PULL           = 13;
    STOP_UPDATES   = 14;
    GET_OBJECT     = 15;
    GET_DATA       = 16;
    DATA_SIGNATURE = 17;
    CANCEL         = 18;
}

// hlavní zpráva, ve které jsou obsaženy další dílčí zprávy
message Message {
    // typ zprávy
    required MessageType   type            =  1;

    // hybridní zprávy
    optional Error         error           =  2;

    // notifikace
    optional Hello         hello           =  3;
```

```
    optional AddedVolume   addedVolume     =  4;
    optional RemovedVolume removedVolume   =  5;
    optional Update        update          =  6;
    optional Synchronized  synchronized    =  7;

    // odpovědi
    optional Success       success         =  8;
    optional Pulled        pulled          =  9;
    optional Snapshot      snapshot        = 10;
    optional Metadata      metadata        = 11;
    optional Data          data            = 12;

    // požadavky
    optional Pull          pull            = 13;
    optional StopUpdates   stopUpdates     = 14;
    optional GetObject     getObject       = 15;
    optional GetData       getData         = 16;
    optional DataSignature dataSignature   = 17;
    optional Cancel        cancel          = 18;
}

// ----------------
// HYBRIDNÍ ZPRÁVY
// ----------------

// chyba
message Error {
    enum ErrorCode {
        INTERNAL_ERROR      =  1;   // chyba na straně uzlu
        SYNTAX_ERROR        =  2;   // chyba v~protokolu, následuje odpojení
        UNKNOWN_MESSAGE     =  3;   // neznámá zpráva, zpráva bude ignorována
        OBJECT_NOT_FOUND    =  4;   // daný objekt nebyl nalezen
        DATA_NOT_FOUND      =  5;   // data nebyla nalezena
        USER_CANCELLED      =  6;   // uživatel zrušil požadavek
        PEER_CANCELLED      =  7;   // transakce ukončena na základě zprávy Cancel
        INVALID_REQ_ID      =  8;   // neexistující transakce
        VOLUME_NOT_FOUND    =  9;   // svazek nebyl nalezen
        BAD_REQUEST         = 10;   // ostatní chyby
    }
    optional uint32     response_to = 1;    // transakční identifikátor (pokud existoval)
    required ErrorCode  code        = 2;    // chybový kód
    optional string     message     = 3;    // dodatečná zpráva
}

// ----------------
// NOTIFIKACE
// ----------------

// povinná úvodní zpráva
message Hello {
    repeated string options         = 1;    // seznam rozšíření
    optional bytes  peer_id         = 2;    // identifikátor uzlu
    optional string peer_name       = 3;    // název uzlu
}
```

```
// dostupný svazek pro synchronizaci
message AddedVolume {
    required bytes  volume_id       = 1;    // identifikátor svazku
    optional string volume_name     = 2;    // název svazku
}

// svazek již není dostupný
message RemovedVolume {
    required bytes volume_id        = 1;    // identifikátor svazku
}

// oznámení o~nových metadatech
message Update {
    required bytes  volume_id       = 1;    // identifikátor svazku
    required bytes  metadata_id     = 2;    // identifikátor metadat
}

// oznámení o~ukončení synchronizace
message Synchronized {
    required bytes  volume_id       = 1;    // identifikátor svazku
    optional bytes  snapshot_id     = 2;    // nepovinný identifikátor posledního snímku
}

// ----------------
// ODPOVĚDI
// ----------------

// odpověd OK
message Success {
    required uint32 response_to     = 1;    // transakční identifikátor
}

// odpověd s~posledním známým snímkem k~danému svazku
message Pulled {
    required uint32 response_to     = 1;    // transakční identifikátor
    repeated bytes  snapshot_ids    = 2;    // identifikátor snímku
}

// objekt typu snímek
message Snapshot {
    required uint32 response_to     = 1;    // transakční identifikátor
    optional bytes  parent_id       = 2;    // nadřazený snímek
    repeated bytes  metadata_ids    = 3;    // identifikátory metadat tohoto snímku
}

// objekt typu metadata
message Metadata {
    enum Type {
        FILE        = 1;    // regulérní soubor
        DIRECTORY   = 2;    // adresář (nemá velikost a~data)
        PSEUDOFILE  = 3;    // další možné rozšíření - speciální případy
    }
    required uint32 response_to     = 1;    // transakční identifikátor
```

```
    optional bytes  parent_id       = 2;    // nadřazená metadata
    optional bytes  resolves        = 3;    // konfliktní metadata
    optional bytes  dir_id          = 4;    // nadřazený adresář
    required Type   type            = 5;    // typ položky v~souborovém systému
    required string file_name       = 6;    // název položky v~UTF-8
    optional uint64 size            = 7;    // velikost souboru v~bajtech
    optional bytes  data_id         = 8;    // obsah souboru (nic v~případě adresáře)
    optional bool   deleted         = 9 [default = false]; // položka smazána
}

// odpověď na zprávu GetData
message Data {
    required uint32 response_to     = 1;    // transakční identifikátor
    required bool   final           = 2;    // poslední zpráva
    required bytes  chunk           = 3;    // data
    // obsah atributu chunk závisí na použité metodě (úplná nebo librsync)
}

// ----------------
// POŽADAVKY
// ----------------

// požadavek na poslední identifkátor snímku svazku, dobrovolně zapnutí oznámení
message Pull {
    required uint32 request_id      = 1;    // transakční identifikátor
    required bytes  volume_id       = 2;    // identifikátor svazku
    optional bool   start_updates   = 3 [default = false];
}

// požadavek na zastavení oznámení o~nových souborech
message StopUpdates {
    required uint32 request_id      = 1;    // transakční identifikátor
    required bytes  volume_id       = 2;    // identifikátor svazku
}

// požadavek na objekt (snímek nebo metadata)
message GetObject {
    required uint32 request_id      = 1;    // transakční identifikátor
    required bytes  object_id       = 2;    // identifikátor metadat nebo snímku
}

// požadavek na objekt data
message GetData {
    enum Method {
        FULL            = 1;    // stahuje vše
        REMOTE_DELTA    = 2;    // použití librsync
        // v~případě REMOTE_DELTA musí následovat zpráva DataSignature
    }
    required uint32 request_id      = 1;    // transakční identifikátor
    required bytes  data_id         = 2;    // identifikátor dat
    required Method method          = 3;    // použitá metoda
}

// tato zpráva následuje za GetData v~případě použití metody REMOTE_DELTA
```

```
message DataSignature {
    required uint32 request_id      = 1;    // transakční identifikátor
    required bool   final           = 2;    // poslední zpráva
    required bytes  chunk           = 3;    // data
}

// zrušení transakce
message Cancel {
    required uint32 request_id      = 1;    // transakční identifikátor
    required uint32 to_cancel       = 2;    // indentifikátor rušené transakce
    // oba identifikátory náleží tomu kdo zprávu poslal
}
```