Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

KIVFS - File Manager pro MS Windows

Plzeň, 2012 Václav Steiner

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 20. června 2012

.....

Václav Steiner

Abstract

KIVFS - File Manager for MS Windows

The primary goal of this bachelor thesis is to design and make MS Windows client for distributed file system called KIVFS. This client is based on its own dynamic link libraries and on the selected file manager which is implemented as a plugin.

Next objective of this work is to compare the cache algorithms. These algoritms have to choose which items, if the cache is full, to discard and which to keep for futher use.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Cílem této práce je v teoretické části obecné seznámení s vlastnostmi distribuovaného souborového systému, algoritmy používanými k odstraňování souborů z cache a s možnostmi klientských implementací pro operační systém MS Windows. Klientská aplikace bude realizována jako zásuvný modul pro vybraného souborového manažera. Toto řešení nevyžaduje narozdíl od jaderného modulu nebo FUSE řešení zásahy do operačního systému a využívá možností používaného souborového manažera.

V praktické části bude vysvětlena konkrétní implementace zásuvného modulu vybraného souborového manažera, včetně využití souboru rozšiřujících knihoven KIVFSTechnology, který implementuje plnou podporu KIVFS.

Řešení je realizováno pro operační systém Microsoft Windows a vyvíjeno v programovacím jazyce C/C++.

2 Distribuovaný souborový systém

Distribuovaný souborový systém[1] je systém umožňující přístup k souborům umístěných na vzdálených serverech, z různých klientských počítačů, pomocí počítačové sítě. Klientské počítače nemají informaci, kde se servery nacházejí či jaké je jejich množství a členění. Mezi servery komunikují na základě stanoveného protokolu. Distribuovaný souborový systém dále umožňuje např. omezení přístupu k datům na základě ACL[1], ověřování identity uživatele, zabezpečený přístup pomocí šifrování apod.

V této kapitole budou, kromě vlastností distribuovaného systému, vysvětleny obecné principy práce se soubory, jejich příjem a odesílání, dále pak replikace a možnosti cache.

2.1 Vlastnosti distribuovaného systému

- **transparentnost** - skrytí faktu uživatelům, že jsou soubory umístěné na vzdálených serverech
- **replikace** - migrace dat mezi servery, uživatelé mohou pracovat i v případě poruchy některého z nich
- **škálovatelnost** - snadná rozšiřitelnost na velké množství strojů
- **bezpečnost** - přenos dat po síti je šifrován
- **nezávislost** - služby distribuovaného systému mohou být poskytovány na různých platformách

2.2 Replikace dat

Distribuovaný souborový systém obsahuje tzv. replikace. Data uživatelů mohou být uloženy v kopiích na více souborových serverech. Výhodou tohoto řešení je dostupnost při výpadku některého ze serveru a možnost rozdělování zátěže za účelem lepší škálovatelnosti. Nevýhodou je problém duplicity, díky níž je nutné zajišťovat synchronizaci všech kopií.

[1] Access Control List

Synchronizací se rozumí vykonání jednotlivých transakcí na všech serverech ve stejném pořadí. Rozlišujeme následující způsoby implementace:

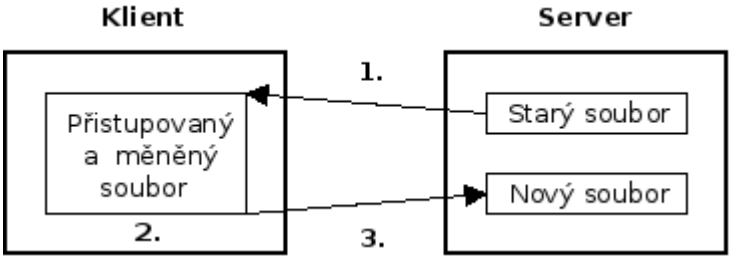
- **synchronní** - změny se provádí ihned na všechny servery
- **asynchronní** - změny se provádí až po určité události

2.3 Přenos souborů obecně

Pro přenos souborů mezi serverem a klientem je možné použít dva modely.

2.3.1 model client/server

Pracuje se s celým souborem, který je přenesen ze serveru na klienta. Po provedení změn je pak nahrán zpět na server (znázorněno na obrázku č. 1). Klient tedy se souborem pracuje lokálně (v paměti nebo na pevném disku), včetně podpory cache. Jedná se o nejjednodušší implementaci.

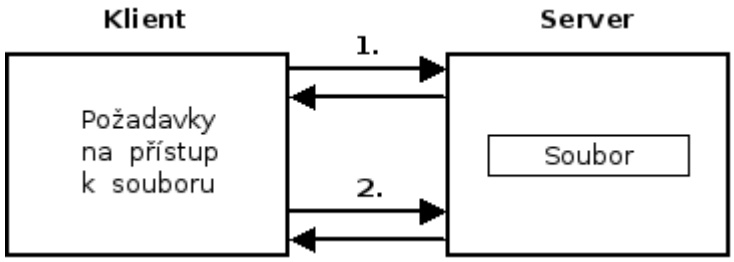


Obrázek 1: Model client/server

2.3.2 model vzdáleného přístupu

Přenášeny jsou pouze bloky souboru, které jsou uživatelem požadovány. Změněné bloky jsou poté nahrány zpět na server (znázorněno na obrázku č. 2). Tento model využívá např. protokol NFS[2], používaný k připojování diskových svazků vzdáleného serveru na klientskou stanici.

[2]Network File System



Obrázek 2: Model vzdáleného přístupu

2.4 Cache

Cache se v distribuovaném systému používá k dočasnému ukládání souborů. Rozlišujeme následující modelové situace použití:

- **systémy bez podpory cache**

  Nejjednodušší případ, není vyžadováno žádné lokální úložiště dat. Důsledkem jsou však opakované přenosy v případě, že uživatel žádá několikrát stejný soubor, který je potřeba přenést znovu.

- **systémy s cache umístěnou v paměti**

  Využití u tzv. bezdiskových klientských stanic. Dosahuje rychlejších přístupů (čtení/zápis) než v případě využití diskové cache, avšak při výpadku klientské stanice dojde ke kompletní ztrátě dat.

- **systémy s cache umístěnou na lokálním disku**

  V tomto případě je nutné mít dostatečnou volnou kapacitu na pevném disku a nebo, což bývá nejčastější řešení, nastavit kvóty tomuto úložišti. Je důležité vždy naimplementovat některou z metod odstraňování souborů z cache, neboť dříve nebo později se tento prostor zaplní. Tomuto tématu se bude později věnovat kapitola 3. V případě výpadku počítačové sítě máme k dispozici lokální kopii do té doby používaných souborů. Tu můžeme použít jako základ pro offline režim, aby měl uživatel přístupné alespoň tyto soubory (prakticky viz. kapitola 7.8).

2.5 Příjem souborů

Při příjmu souborů, za použití cache, je samozřejmě nutné řešit aktuálnost uchovávaných souborů, abychom neposkytovali uživateli soubor z cache, který je již zastaralý. Aktuálnost požadovaného souboru můžeme zkontrolovat např. porovnáním získáné hodnoty `version`[3] z metadat poskytnutých serverem a poslední lokálně uložené hodnoty v cache.

Může nastat situace, že uchováváme v cache soubor s hodnotou `version`, který jiný uživatel na serveru smaže a vytvoří se stejným názvem. Avšak s naprosto jiným obsahem. Proto by bylo vhodné provádět porovnání ještě proti další hodnotě. Tou by mohla být hodnota `creation_time`[4]. Tím zajístíme, že skutečně soubor který požadujeme, nebyl znovu vytvořen.

Rozlišujeme následující případy kontroly konzistence:

- **při každém požadavku** - ke kontrole dochází vždy při komunikaci se serverem. Je tedy zajištěno, že jsou v cache uchovány stále aktuální všechny soubory. Dochází zde však k přenosům souborových metadat i v případě, že přenos souborů uživatel nevyžaduje.

- **pouze při otevření souboru** - ke kontrole dochází až při skutečném otevřením souboru před jeho přenosem. Nedochází zde k nadbytečným přenosům souborových metadat a zaktualizuje se pouze požadovaný soubor.

V KIVFS implementaci je použit případ č. 2, tedy kontrola konzistence pouze při otevření souboru.

2.6 Odesílání souborů

Při odesílání souborů rozlišujeme následující možnosti zápisu na server:

- **přímý zápis** - všechny změny jsou na server odeslány okamžitě.

[3] číselná hodnota aktuální verze souboru

[4] čas vytvoření souboru

- **zpožděný zápis** - změny jsou na server odeslány až po určitém časovém období. Dochází zde k omezení přenosu dočasných souborů, které jsou zapsány, přečteny nebo smazany ještě předtím, než se odešlou na server. Tato metoda bohužel nepřináší přílišnou spolehlivost, protože v případě poruchy stroje mezi stanovenými synchronizačními intervaly může dojít ke ztrátě dat.

- **zápis po uzavření souboru** - změny jsou na server odeslány až ve chvíli, kdy dojde k uzavření souboru. Toto je výhodné za předpokladu, že je soubor otevřen v režimu append a probíhá do něj zápis. Stejně jako u předchozí metody zde dochází při poruše systému ke ztrátě dat. Tato metoda je použita v KIVFS implementaci pro svoji jednoduchost a také z důvodu, že pro práci se soubory je využíván model klient/server (viz kapitola 2.3.1).

3 Algoritmy pro odstraňování souborů z cache

V kapitole o distribuovaných systémech jsme cache probrali z obecného hlediska fungování. V této se zaměříme na oblast odstraňovaní souborů z cache, protože lokální souborové úložiště je vhodné omezit na určitou velikost a následně pak zvolit vhodnou metodou/algoritmus[2] pro odstraňování souborů. Dále pak na měření úspěšnosti algoritmů z pohledu počtu nalezených souborů v cache a procentuálního vyjádření ušetřených přenosů.

3.1 Random

Jedná se jednu ze základních a nejjednodušších metod. Z cache se náhodně vybere soubor, který bude odstraněn. Výhodou je triviální implementace, ale vzhledem ke skutečně náhodnému výběru souborů může často dojít k odstranění používaných souborů.

3.2 First-In, First-Out - FIFO

Tato metoda odstraní nejstarší uložený soubor. Nejstarším souborem je zde myšlen soubor podle příchodu do cache. Jedná se tedy o variaci na klasickou implementaci abstraktního datového typu "fronta". Programová implementace je taktéž jednoduchá, ovšem i nejstarší uložený soubor může být často využíván uživatelem. Tato metoda proto také není příliš efektivní.

3.3 Least Recently Used - LRU

Odstraňovat se bude nejdéle nepoužívaný soubor z cache. K jeho zjištění se využívá časového razítka, tedy informace o souboru, kdy byl naposledy požadován. Odstraněn bude tedy ten soubor, který je vyhodnocen jako nejstarší.

Tato implementace však vyžaduje náročnější udržování informací o uložených souborech v cache, neboť je nutné postupně zkontrolovat všechny soubory a porovnat jednotlivá

časová razítka. Při použití databázové implementace lze dosáhnout díky optimalizovaným SQL[3] dotazům rychlejších výsledků oproti implementaci např. zásobníkem.

3.4 Least Frequently Used - LFU

Odstraňovat se bude nejméně často používaný soubor. U každého souboru v cache se uchovává hodnota, kolikrát byl daný soubor přistupován, požadován. Této hodnotě říkáme počet hitů. Odstraní se ten soubor, který bude mít tuto hodnotu nejmenší. Vyskytne-li se několik souborů se stejným, tedy nejmenším, počtem hitů, aplikuje se na tento výběr ještě metoda LRU.

Podobně jako u metody LRU zde dochází k náročnějšímu udržování informací o uložených souborech v cache. Výkonnostně lze metodu LFU srovnávat s metodou LRU, neboť fungují na podobném principu, avšak podle všeobecně známých studií je metoda LRU úspěšnější ve výběru nepoužívaných souborů.

3.5 LFU s hity ze serveru

Jedná se o klasickou implementaci metody LFU s tím rozdílem, že se prvotní počet hitů získá v rámci metadat (informací o souboru) ze serveru. Nový soubor v cache nebude mít tedy hodnotu 1, jako by tomu bylo u předchozí metody. Počet hitů získaný ze serveru tedy reflektuje žádanost daného souboru na základě jeho oblíbenosti i ze strany ostatních uživatelů. Rozšířená metoda LFU s hity ze serveru je tedy úspěšnější ve výběru nepoužívaných souborů než-li klasická, ze které vychází.

3.6 LRU & LFU s hity ze serveru

Hybridní metoda spojující výhody metod LRU a LFU, navržená (podobně jako předchozí metoda) na KIV[4], se snaží vyřešit situaci, kdy nechceme v cache zbytečně uchovávat soubory, které mají vysoký počet hitů na zápis a nízký na čtení (např. logy). Dále pak soubory, které sice mají vysoký počet hitů, ale již dlouho nebyly přistupovány. Prvotní počet hitů získáme opět v rámci metadat ze serveru.

Na základě výpočtů uchováváme pro každý soubor dvě 16-bitová nezáporná čísla - prioritu P_{LRU} a prioritu P_{LFU}. Tyto hodnoty poté vynásobíme předem zadaným koeficientem (K_1, K_2) a sečteme. Pomocí těchto koeficientů jsme schopni upřednostnit jednu metodu před druhou. Koeficienty typicky uvádíme jako poměr hodnot, např. 1:2.

3.6.1 Výpočet priority LRU

U každého souboru uchováváme informaci, kdy byl naposledy přistupován - časový otisk, nejlépe v ns. Zároveň udržujeme informaci o nejdéle nepoužívaném souboru. Při příchodu nového souboru či při jeho opětovném přístupu, aktualizujeme prioritu u všech uložených souborů v cache pomocí lineární interpolace.

3.6.2 Výpočet priority LFU

Při příchodu nového souboru do cache budeme provádět výpočet priority na základě získaných metadat ze serveru (počet hitů pro čtení/zápis, celkový počet souborů) a také zohledníme, zda dochází častěji k zápisu do souboru. Takové soubory kvůli zachování konzistence dat v cache nechceme.

3.7 Měření efektivity cache algoritmů

Pro zjištění nejefektivnější metody odstraňování souborů z cache bylo provedeno několik měření. Měření bylo provozováno na operačním systému Microsoft Windows XP na file systému NTFS[5]. Pro potřeby měření bylo náhodně vygenerováno 500 souborů o velikosti 0,5MB - 4MB, které byly číselně pojmenovány, tedy ”1”, ”2”, ... ”500”. Dále pak byla pro každý soubor vygenerována jeho metadata (čas, počet hitů, hodnota verze apod.).

Měření probíhalo na základě dvou přístupů (náhodný a prioritní) a měřil se vždy vzorek 10 000 souborů na různě zadaných velikostí cache, tedy 8, 16, 32, 64, 128 a 256MB, a měřila se hodnota HIT a MIS. Dále velikost přenesených a ušetřených dat díky nálezu v cache. V měření byly použity všechny výše jmenované algoritmy pro odstraňování souborů z cache.

[5]New Technology File System - souborový systém vyvinutý pro OS Windows řady NT

3.7.1 Náhodný přístup

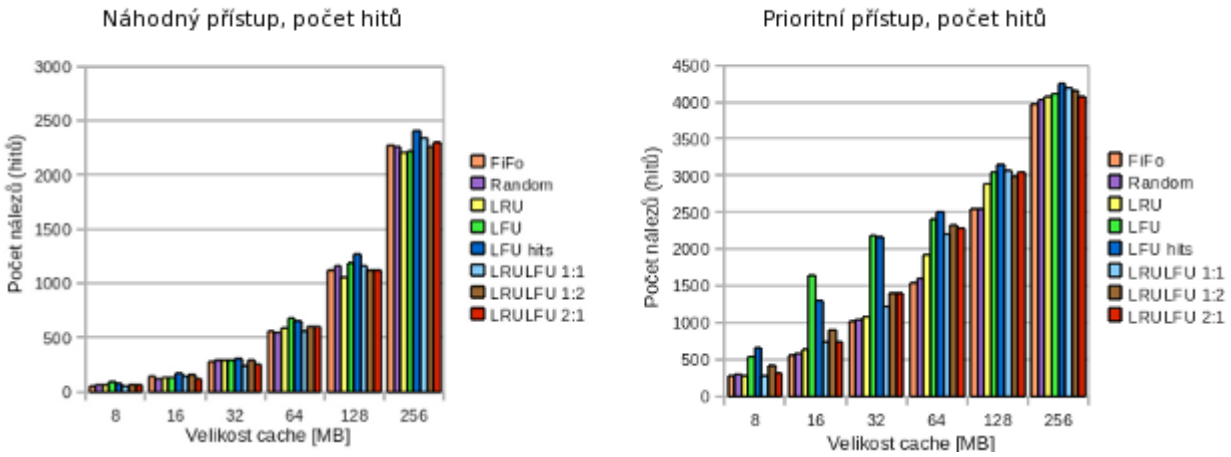
Pomocí generátoru náhodných čísel dojde k vygenerování názvu souboru, který se bude přistupovat.

3.7.2 Prioritní přístup

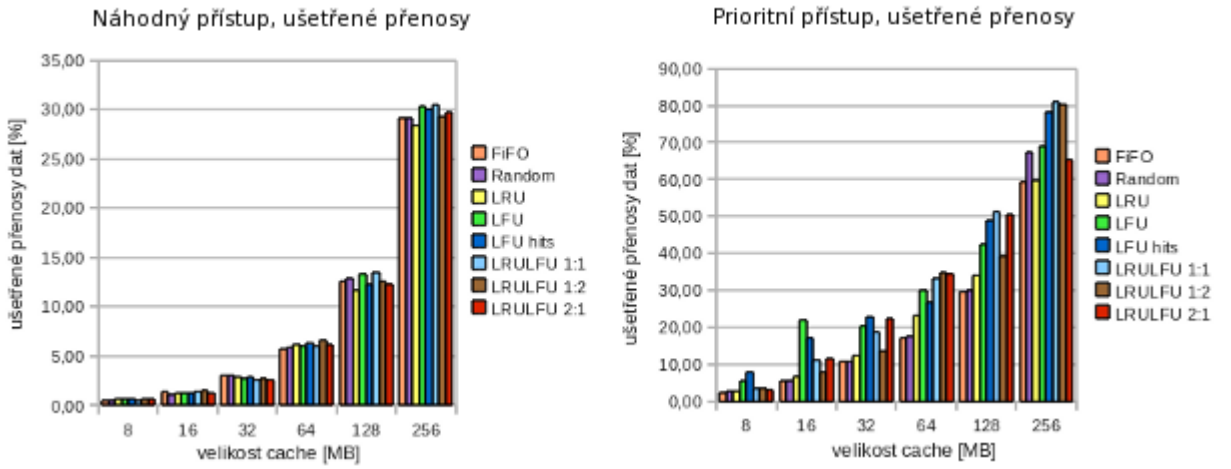
Zde se také využije generátoru náhodných čísel k určení názvu souboru, ale některé soubory budou preferované. Pomocí operace modulo 5 se provede dělení získané číselné hodnoty. Na základě výsledku beze zbytku se bude požadovat jiný soubor. Např. místo názvu souboru ”5”, ”10”, ..., ”95” se přistoupí soubor ”100”. Místo ”105”, ”110”, ..., ”195” soubor ”200” apod.

3.7.3 Výsledky měření

Výsledky úspěšnosti daných algoritmů lze porovnat na následujících grafech (obr. č. 3 a č. 4). Souhrnné tabulky jsou uvedeny v příloze A.1 (tabulky č. 7, 8, 9, 10).



Obrázek 3: Měření efektivity cache algoritmů - počet HITů



Obrázek 4: Měření efektivity cache algoritmů - ušetřené přenosy

3.8 Zhodnocení výsledků

Na základě naměřených výsledků vidíme, že při odstraňování souborů, při hraniční velikosti cache, dosahuje obecně nejlepších výsledků vlastní navržená metoda LFU s hity ze serveru a klasická metoda LFU. Dle uveřejněných a všeobecně známých studií, by měla být metoda LRU o 3-5% úspěšnější oproti LFU. Mnou naměřený propad mohl být způsoben principem volby přistupovaných souborů.

Podle výše uvedených grafů (obr. č. 3 a č. 4) a výsledků uvedených v příloze (tabulky č. 7, 8, 9, 10) vidíme (a zajímavější je to právě v případě ušetřených přenosů prioritního přístupu), že pro větší velikost cache úložiště se dostává do popředí hybridní algoritmus LRU&LFU v závislosti na zvolených prioritách a to na úkor klasických metod.

Předmětem měření bylo porovnání úspěšnosti v odstraňování nepoužívaných souborů jednotlivých metod, co do počtů HITů (=počtu úspěšných nálezů) a co do procentuální úspěšnosti ušetřených síťových přenosů díky uchovaným souborům v cache. Na základě tohoto je pak možné zvolit použití té nejvhodnější metody.

4 Souboroví manažeři

Cílem této kapitoly je shrnutí možností aktuálně dostupných souborových manažerů a jejich vlastností pro použití plné funcionality KIVFS serverů. Porovnáme komerční a nekomerční zástupce a zjistíme, zda-li jsou vhodnými kandidáty pro implementaci zásuvného modulu (pluginu).

4.1 Komerční zástupci

Komerční souboroví manažeři, jak vyplývá z názvu, jsou ve většině případů profesionální aplikace, které jsou šířeny pod licencemi určenými k prodeji díla. Většinou jsou distribuovány jako shareware. Uživatel má možnost si aplikaci vyzkoušet a po určité době ji zakoupit nebo odinstalovat.

4.1.1 Total Commander

Total Commander[5], dříve Windows Commander, je v současné době jeden z nejpopulárnějších komerčních souborových manažerů pro Microsoft Windows[6] a nově i pro Android[7]. V době psaní této práce je dostupný pro MS Windows ve verzi 7.57a a 8.0 beta 25. Pro Android pak ve verzi 1.0 rc6. Je lokalizovaný do velkého množství jazyků, včetně češtiny. Díky podpoře a nepřeberného množství různých typů zásuvných modulů (souborové, archivační aj.) se jedná opravdu o špičku ve svém oboru.

Pro tvorbu zásuvných modulů disponuje vlastním SDK[6], kterým se inspirovali i další alternativní zástupci, i z řad nekomerčních variant. Licence je ve variantě shareware, doba vyzkoušení je stanovena na 30 dnů.

4.1.2 Altap Salamander

Altap Salamander[8] pochází z dílny českých autorů. V minulosti byl znám pod názvem Servant Salamander. Česká lokalizace tedy ve výčtu podporovaných jazyků nemůže chybět.

[6]Software Development Kit

Momentálně je ke stažení verze 2.54. Podobně jako Total Commander disponuje vlastním SDK pro tvorbu vlastních zásuvných modulů, které je dokonce o něco více obsáhlejší nežli konkurence. Spolu s výše zmiňovaným Total Commanderem patří k nejpoužívanějším souborovým správcům. Největší jeho předností je velké množství zabudovaných zásuvných modulů v základní instalaci.

Pro kompatibilitu s Total Commanderem je možné doinstalovat zásuvný modul *TCProxy*, který umožňuje import zásuvných modulů určených právě pro tohoto oblíbeného souborového manažera. Jelikož je však rozumější naprogramovat nativní zásuvný modul a využít tak všech nabízených možností tohoto prostředí, skýtá se tak alespoň další možná cesta, kterou se v budoucnu vydat.

Stejně jako u Total Commander má uživatel možnost vyzkoušet si produkt po 30 dní a poté by měl v případě dalšího užívání zaplatit licenční poplatek.

4.1.3 Speed Commander

Jedná se o aplikaci pocházející z Německa. V současné době Speed Commander[9] existuje ve verzi 14.20. Mezi její výhody bezesporu patří nativní podpora 64bit archtektury a další drobnosti, které konkurence neobsahuje. Například malý webový prohlížeč, podpora maker pro zautomatizování periodicky se opakujích úkonů apod. Podpora zásuvných modulů (v jejich pojetí se jedná o Add-Ins) existuje, ale SDK pro tvorbu vlastních modulů ne. Existuje však Add-Ins *WfxWrapper*, který umožnuje použití souborových zásuvných modulů primárně určených pro Total Commander. Bohužel jeho funkčnost není ideální, zkušební zásuvný modul se nanačetl korektně. Dalším nedostatkem je lokalizace. K dispozici je kromě německého překladu také anglický, který však není zcela kompletní.

Tento produkt je také licencován jako shareware a je tak možné jej využívat 60 dní.

4.1.4 EF Commander

Jde o produkt firmy EF software[10], pocházející taktéž z Německa. Firma, která stojí za vývojem tohoto souborového manažera vyvíjí ještě i další produkty např. EF Startup

Manager[11], EF Mailbox Manager[12] apod. Kromě komerční verze, momentálně 8.40, je k dispozici i verze Free a Lite, o kterých se zmíním v kapitole 4.2.4. Jako zajímavost je možné uvést, že počátky vývoje sahají až do roku 1994, kdy byl tento produkt provozován na operačním systému IBM OS/2[13], pod názvem Presentation Manager, jako náhrada produktu Norton Commander pro prostředí MS DOS. Od roku 1996 je portován pro MS Windows.

Stejně jako většina aplikací podporuje také Total Commander API, tudíž je možné využít všechny typy jeho zásuvných modulů. Stejně jako u předchozího zástupce, zkušební zásuvný modul nefungoval správně. Lokalizace je dostupná i v češtině a stejně jako konkurence obsahuje rozmanité množství zásuvných modulů, např. přímé spojení se zařízeními PDA na platformě Windows CE[14].

Licencování je opět řešeno jako shareware se zkušební dobou 30 dní.

4.2 Nekomerční zástupci

Nekomerční zástupci souborovích manažerů jsou většinou vyvíjeni za účelem existence ekvivalentu (často s omezenou funkčností) k výše uvedeným zástupcům v bezplatné freeware licenci. Program je tedy možné bezplatně používat a šířit, autor si však ponechává autorská práva, nedovoluje například program upravovat a může omezit používání pouze pro specifické účely (např. omezení použití ve firemním prostředí).

4.2.1 Free Commander

Free Commander[15] je jednou ze zajímavých alternativ ke komerčním řešením. Momentálně je možné jej stáhnout ve verzi 2009.02b. To nejdůležitější co nás zajímá - zásuvné moduly - nepodporuje. Na webu je přítomnost různých zásuvných modulů (otázkou je zda i vlastních) slibována s příštím vydáním, ale vzhledem k ne moc častým aktualizacím (poslední stabilní verze vyšla v září roku 2010) není jisté, kdy tato verze bude vlastně dostupná.

S ohledem na absenci podpory zásuvných modulů je tento souborový manažer pro KIVFS implementaci nepoužitelný.

4.2.2 Unreal Commander

Unreal Commander[16], aktuálně ve verzi 0.96, lze srovávat s Free Commanderem. Oproti němu disponuje rozšířením o zásuvné moduly, ale pouze o typy WLX[7] a WCX[8]. Vlastní SDK prostředí pro tvorbu pluginů nemá, lze používat již hotové primárně určené pro Total Commander. Pokud pomineme místy poněkud zvláštní uživatelské rozhraní (samozřejmě lze přizpůsobit), je zde další trochu nepříjemná vlastnost. Tou je nutnost registrace programu, jinak je uživatel stále upozorňován na chybějící licenční klíč. Co se týká lokalizace, češtinu je možné stáhnout a nainstalovat dodatečně.

Tento souborový manažer je tedy pro možnost implementace KIVFS klienta taktéž nepoužitelný. Nepodporuje zásuvný modul pro přístup souborovým systémům.

4.2.3 Double Commander

Jedná se o opensource, multiplatformní souborový manažer. Je tedy možné jej provozovat na MS Windows i Linuxu. Momentálně je Double Commander[17] dostupný ve verzi 0.5.4 v různých lokalizacích, včetně češtiny. Je rychlý, jednoduchý a přitom mu nechybí potřebná funcionalita pro běžné užívání.

Další zajímavostí je, že využívá API Total Commanderu (rozdílné jsou pouze popisy hlavičkových funkcí) a podporuje tak všechny jeho typy pluginů.

4.2.4 EF Commander Free

Nekomerční varianta komerčního EF Commanderu[10]. Jedná se o tak omezenou verze, že jeho funkčnost lze přiřadit na úroveň běžného Windows průzkumníka. Přizpůsobit nelze skoro nic a verze umožňuje pouze běžné operace se soubory - vytváření, kopírování, mazání - byt' s komfortem dvoupanelového rozhraní.

[7] zobrazovací moduly - např. prohlížeče obrázků

[8] komprimační moduly

4.2.5 Fine Commnader

Fine Commander[18] je souborový manažer vyvíjený na Katedře informatiky a výpočetní techniky ZČU, je aktuálně dostupný ve verzi 2.0 a provozován nad .NET prostředím. Zásuvné moduly však nepodporuje.

4.3 Shrnutí

Porovnáním uvedených souborových manažerů je patrné, že v této oblasti je stále lídrem Total Commander. Inspirovali se jím i ostatní kandidáti a umožňují dokonce i nativní použití zásuvných modulů ve svých produktech. Mírně se však odlišuje produkt Altap Salamander, který navíc nabízí vlastní SDK pro tvorbu zásuvnách modulů s malými rozdíly a rozšířeními oproti svému největšímu soupeři. Zajímavým počinem je také nekomerční produkt Double Commander, který nejen že plně podporuje API Total Commanderu (s upravenými hlavičkovými funkcemi), je možné jej provozovat i na platformě Linux.

V tabulce č. 1 je znázorněno jednoduché shrnutí, kteří zástupci umožňují implementovat KIVFS funcionalitu jako zásuvný modul.

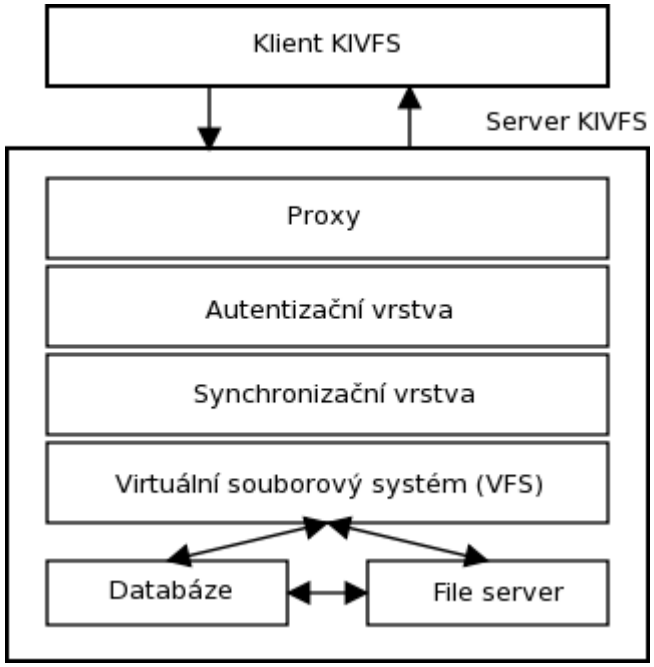
Vzhledem ke své největší rozšířenosti a kompletní podpoře zásuvných modulů byl vybrán souborový manažer Total Commander. Pro něj bude implementována plná podpora KIVFS, která je na operačním systému MS Windows v současné době realizována jako knihovna KIVFSTechnology (viz. kapitola 6.1).

Komerční zástupci		
Název produktu	**Cena**	**Podpora KIVFS**
Total Commander	$46.00	X
Altap Salamander	$29.95	X
Speed Commander	$56.50	
EF Commander	$37.50	
Freeware zástupci		
Free Commander	$0	
Unreal Commander	$0	
Double Commander	$0	X
EF Commander Free	$0	
Fine Commnader	$0	

Tabulka 1: Porovnání souborových manažerů

5 Projekt KIVFS

KIVFS[20] je distribuovaný file systém, navržený na Katedře informatiky a výpočetní techniky ZČU v Plzni. Cílem tohoto projektu bylo navrhnout a implementovat DFS a současně odstranit nedostatky, které známé implementace (jako např. AFS, CODA apod.) obsahují. Projekt je rozdělen do dvou částí. První jsou klientské implementace, vyvíjené pro různá prostředí a platformy a druhou je serverová implementace (OS Linux), která se dále dělí do dalších vrstev, viz obrázek č. 5:



Obrázek 5: Schema součástí KIVFS projektu

Základní popis jednotlivých vrstev serverové implementace:

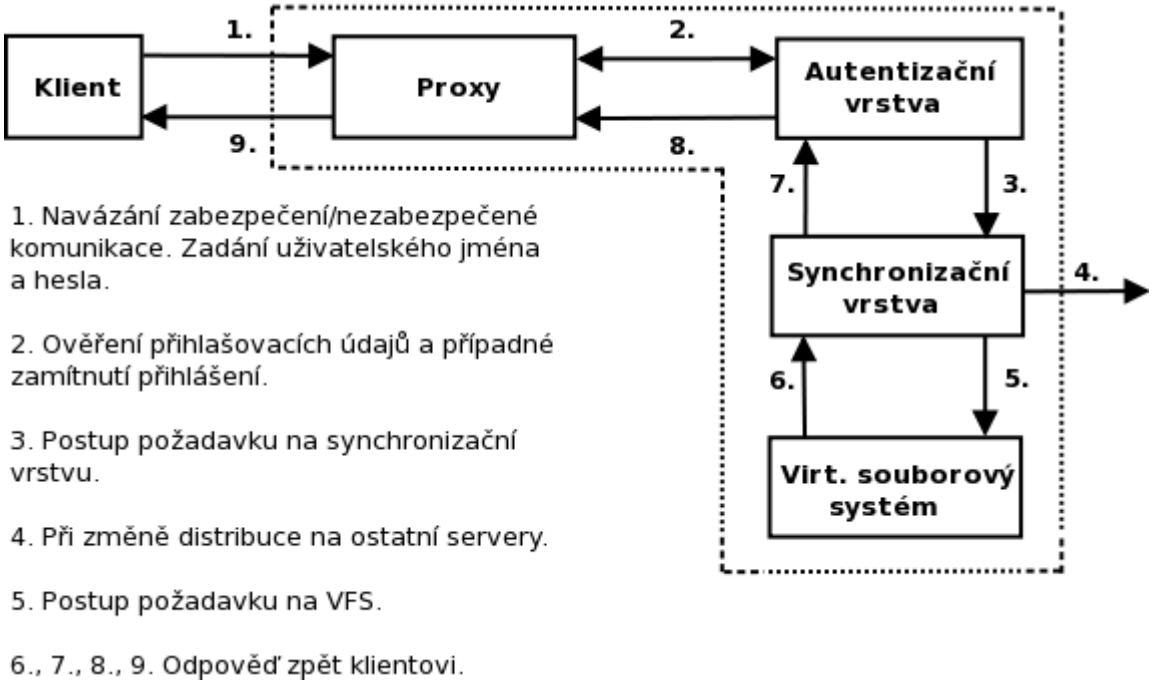
- **Proxy** - inicializuje komunikaci (zabezpečenou či nezabezpečenou) s KIVFS klientem a přenáší požadavky do ostatních vrstev.
- **Autentizační vrstva** - ověřuje uživatele na základě jména a hesla a kontroluje, zda má uživatel přístup ke KIVFS serverům.

- **Synchronizační vrstva** - provádí synchronizaci dat mezi ostatními KIVFS servery.
- **Virtuální souborový systém** - pracuje s fyzickými soubory a jejich metadaty. Dělí na databázi a file server, které soubory poskytují uživateli.

Podrobnější popis fungování serverové implementace je uveden v bakalářské práci KIVFS: Souborový a databázový server[21].

5.1 Princip komunikace

Princip komunikace mezi jednotlivými vrstvami je naznačen na následující ukázce. Jedná se o navázání spojení k serveru (šifrování je realizováno pomocí OpenSSL viz kapitola 5.1.1), zadání přihlašovacích údajů (realizováno mechanismem Kerberos viz kapitola 5.1.2) a vykonání požadavku např. vytvoření nového adresáře (viz obrázek č. 6):



Obrázek 6: Princip komunikace v KIVFS

5.1.1 OpenSSL

Pro šifrování přenosů je v KIVFS využita implementace OpenSSL[22]. Ta je aktuálně dostupná ve verzi 1.0.1b. OpenSSL poskytuje knihovny napsané v jazyce C, které zahrnují základní kryptografické funkce. KIVFS protokol využívá kryptografický algoritmus `RC4-MD5`.

Popis fungování OpenSSL je podrobně uvedeno v bakalářské práci KIVFS: Zabezpečení, šifrování a ověření identity[23].

5.1.2 Kerberos

Ověřování uživatelů, kteří jsou oprávněni pracovat se systémem, je realizováno pomocí autentizačního protokolu Kerberos[24]. Pracuje na principu ticketů, které slouží k ověření identity uživatelů. Pro svoji činnost potřebuje důvěryhodnou třetí stranu, tzv. Key Distribution Center (KDC), sestávající se ze dvou oddělených částí: Autentikačního serveru (AS) a Ticket-Granting serveru (TGS). Hlavní výhodou Kerberosu tedy je, že se heslo nikdy nepřenáší sítí.

Existují dvě implementace Kerberosu, Heimdal a MIT. V KIVFS je využita implementace MIT z důvodu bezproblémové funkčnosti i pro klienty v MS Windows.

Popis fungování Kerberosu je podrobně uvedeno také v bakalářské práci KIVFS: Zabezpečení, šifrování a ověření identity[23].

5.2 Komunikační protokol

KIVFS komunikuje pomocí komunikačního protokolu, fungující nad TCP spojením. Rozdělujeme jej na dvě části, hlavičku a data. Jedná se o datové bloky obsahující zabalená[9] data, které si klient a server vzájemně vymněňují.

[9]způsob "zabalování" dat je popsán v kapitole 6.5

5.2.1 Hlavička

Hlavička je základním stavebním prvkem veškeré komunikace a jedná se o úvodní datový paket, kterým si protistrany vyměnují informace o úkonech, které se mají provést. Při každém požadavku o provedení úkonu na straně serveru sestaví klient hlavičku (její struktura je uvedena v tabulce č. 2) obsahující položky podle nadefinované struktury a pošle je sítí.

název pole	**typ**	**popis**
`magicnumber`	`unsigned __int32`	`magic number`
`timestamp`	`unsigned __int64`	`časové razítko`
`command`	`unsigned __int32`	`příkaz`
`return_code`	`unsigned __int32`	`návratový kód`
`data_size`	`unsigned __int64`	`velikost datového bloku`
`session`	`unsigned __int64`	`session id (pouze pro autentizační vrstvu)`
`type`	`unsigned __int32`	`typ (pouze pro autentizační vrstvu)`

Tabulka 2: Struktura hlavičky

Nejdůležitejší položky jsou `command`[10], `return_code`[11] a `data_size`[12].

5.2.2 Datový paket

Datový paket obsahuje již zmíněné zabalené informace, které jsou úzce svázané s hlavičkou a jsou závislé na zvoleném příkazu. Požadujeme-li od serveru např. vytvoření adresáře, obsahem je tedy logicky absolutní cesta k adresáři včetně jeho názvu.

Pro každý příkaz je struktura různá, avšak obě strany tuto definici znají. V zásadě rozlišujeme dvojí "zabalování" dat. V případě, že protistraně posíláme pouze číselné hodnoty, je stavba datového paketu velice jednoduchá. Blok obsahuje pouze číslené hodnoty, v závislosti na jejich typu (viz. tabulka č. 3):

[10] seznam nejdůležitějších příkazů je uveden v příloze A.2

[11] používá se pouze za předpokladu, že hlavička slouží jako odpověď pro daný požadavek

[12] udává velikost následného datového bloku

hodnota __int64	hodnota __int32	hodnota __int16	hodnota __int8

Tabulka 3: Struktura datového paketu s číslenými hodnotami

Pokud posíláme kromě číselných hodnot také řetězec, je potřeba jej poslat s informací o jeho délce. Ta se uloží do 32bit číselné hodnoty a až poté se vloží obsah řetězce (viz tabulka č. 4):

délka řetězce	hodnota řetězce	délka řetězce	hodnota řetězce

Tabulka 4: Struktura datového paketu s řetězci

Příklad pro sestavení `open_file`. `KIVFS_FILE_MODE_READ` je konstanta, v našem případě s hodnotou 1. Tento datový paket má tedy velikost 24 bajtů (viz tabulka č. 5):

délka řetězce	absolutní cesta	režim přenosu
16	/tmp/sestava.txt	KIVFS_FILE_MODE_READ

Tabulka 5: Příklad datového paketu open_file

Datový paket se neočekává v případě, kdy je potřeba odeslat pouze hlavičku. Typickým příkladem je odpověď serveru na provedené příkazy, či různá potvrzení (např. o informace dokončení přenosu souboru). Hodnota `data_size` v hlavičce je tedy nulová a informuje, že protistrana data očekávat nemá.

6 KIVFSTechnology

6.1 Úvod

KIVFSTechnology je soubor dynamických knihoven DLL (Dynamic-link library), implementovaných v jazyce C/C++[25] ve vývojovém prostředí Microsoft Visual Studio 2010[26]. Jedná se o práci nad rámec zadání, kterou jsem implementoval kvůli zjednodušení vývoje dalších klientských aplikací pro MS Windows a obsahuje tedy veškerou funcionalitu KIVFS.

V počátku fungování KIVFS projektu probíhal vlastně veškerý vývoj klientské části výhradně v zásuvném modulu pro Total Commander. Při dalších implementacích klientů pro MS Windows, např. klient s nativní podporou ve formě IFS (Installable File System), by se muselo jít cestou buď náročného opisování stávajího kódu nebo programováním nového kódu. Proto se přistoupilo k oddělení veškeré programové logiky a implementace týkající se KIVFS a vložení do dynamických DLL knihoven. Výsledkem je sice náročnější vývoj a debugování nových funkcionalit, ale vše se vyvíjí pouze jednou a následně několikrát použije. Tvůrce nového klienta má k dispozici aktuální popis (referenční příručku) všech funkcí v PDF souboru (viz příloha na CD) a prostým voláním jednotlivých funkcí, které potřebuje, je schopen za velmi krátký čas implementovat plnohodného klienta. Další výhodou jsou také změny ve vývoji. Stačí zakualizovat pouze konkrétní knihovní soubor.

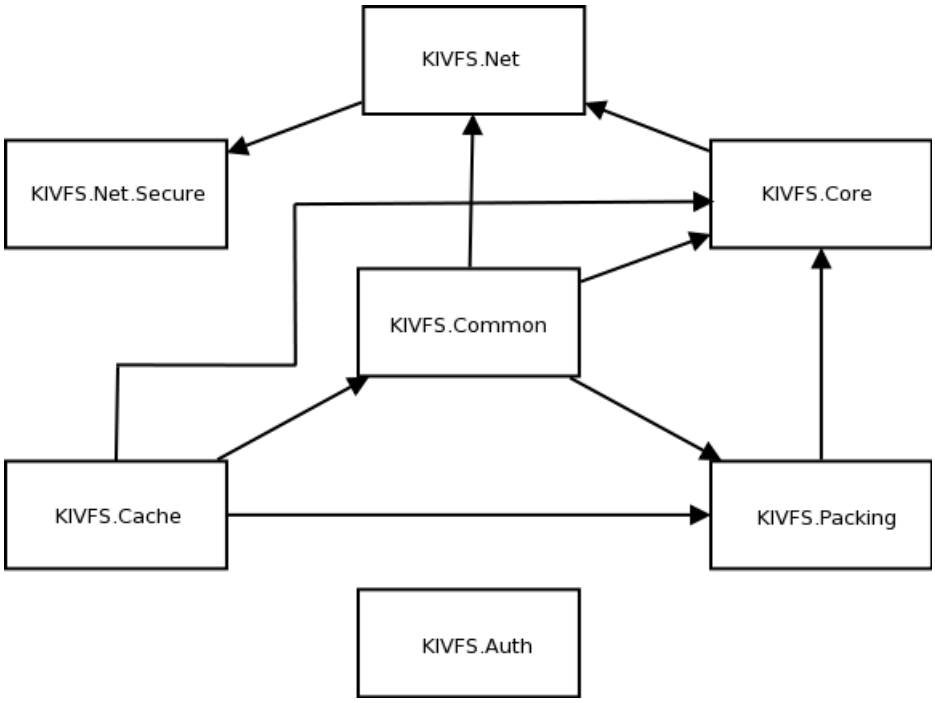
KIVFSTechnology je soubor dynamických knihoven DLL. Všechny funkce jsou tedy uloženy tématicky do více projektů. Pro jednoduchost je nazývejme moduly. Některé z nich jsou pro finální funkčnost volitelné, např. cache modul, ostatní povinné, protože je většina z nich mezi sebou provázaná.

Jedná se tyto moduly:

- **KIVFS.Core** - obecná funkčnost komunikačního protokolu KIVFS, viz kapitola 6.2
- **KIVFS.Net** - podpora síťové komunikace, viz kapitola 6.3
- **KIVFS.Net.Secure** - rozšíření předchozího modulu o šifrovací mechanismum, viz kapitola 6.4

- **KIVFS.Packing** - příprava a čtení datových paketů, viz kapitola 6.5
- **KIVFS.Common** - finální funkce pro konkrétní účel, viz kapitola 6.6
- **KIVFS.Auth** - ověřování jména a hesla vůči autentizační vrstvě, viz kapitola 6.7
- **KIVFS.Cache** - podpora cache, viz kapitola 6.8

Závislosti jednotlivých modulů ukazuje obrázek č. 7:



Obrázek 7: Závislosti modulů KIVFSTechnology

6.2 KIVFS.Core

Modul KIVFS.Core je nejdůležitější. Obsahuje totiž fundamentální funkce pro obecnou funkčnost komunikačního protokolu KIVFS. Jedná se tedy o definici struktur hlavičky, souborových metadat a jejich obslužných funkcí, dále pak seznam chybových stavů a příkazů, určující požadované operace[13]. Je to tedy základ všech ostatních modulů.

Souborová metadata (viz tabulka č. 6) slouží k ukládání všech poskytovaných informací o souborech v adresáři, který uživatel v rámci listování adresářové struktury vyžaduje. Jelikož očekáváme více souborů, jsou všechna metadata ukládána do spojeného seznamu, ze kterého mohou být pak následně jednoduše přečtena.

název pole	**typ**	**popis**
filetype	unsigned __int32	typ - soubor nebo adresář
name	char[255]	název souboru
owner	char[255]	vlastník souboru
group	char[255]	skupina souboru
size	unsigned __int64	velikost souboru
mtime	unsigned __int64	čas změny soubory
atime	unsigned __int64	čas přístupu k souboru
version	unsigned __int64	verze souboru
read_hits	unsigned __int64	počet hitů pro čtení souboru
write_hits	unsigned __int64	počet hitů pro zápis souboru
owner_rights	unsigned __int32	práva vlastníka pro soubor
group_rights	unsigned __int32	práva skupiny pro soubor
others_rights	unsigned __int32	práva ostatních pro soubor
next	kivfs_core_filedata*	spoj pro spojový seznam

Tabulka 6: Struktura souborých metadat

Struktura hlavičky a její funkce již byla popsána v kapitole 5.2.1, proto na tomto místě uvedu pouze hlavičky nejdůležitějších funkcí, které s hlavičkou operují.

```
/* Create KIVFS head */
KIVFSCORE int kivfs_core_create_head(unsigned __int64 timestamp,
    unsigned __int32 command, unsigned __int32 return_code,
```

[13] seznam nejdůležitějších příkazů je uveden v příloze A.2

```
    unsigned __int64 data_size, struct kivfs_core_head *head);

/* Receive KIVFS head */
KIVFSCORE int kivfs_core_recv_head(struct kivfs_core_head *head, int socketfd);

/* Send KIVFS head */
KIVFSCORE int kivfs_core_send_head(struct kivfs_core_head *head, int socketfd);
```

6.3 KIVFS.Net

Tento modul zastřešuje celou síťovou komunikaci, tedy funkce pro připojení a odpojení od serveru, odesílání a příjem datových bloků a seznam aktivních síťových spojení (socketů). Podpora více (alespoň 2) socketů je zde z důvodu, že prvním spojením klient a server mezi sebou pouze komunikují, provádí specifické příkazy, definují způsoby přenosů dat apod. Tím druhým pak přenáší soubory. Je tedy možné odsunout přenos souborů na pozadí a dále s aplikací pracovat.

Pro upřesnění jen uvedu výše zmiňované funkce:

```
/* Connect to KIVFS server */
/* crypt - type of encrypt: openssl/none */
KIVFSNET int kivfs_net_connect(int socketnumber, char *address, int port, char *crypt);

/* Disconnect from KIVFS server */
KIVFSNET int kivfs_net_disconnect(int socketnumber);

/* Send data over socket */
/* data - datablock from transfer */
KIVFSNET unsigned __int64 kivfs_net_send_data(int datasize, void *data, int socketfd);

/* Receive data over socket */
KIVFSNET unsigned __int64 kivfs_net_recv_data(int datasize, void *data, int socketfd);
```

Z výše uvedených funkcí je patrné, že se doporučuje použití vlastních funkcích pro odesílání a příjem dat oproti běžným `recv()` a `send()`. Důvodem je, že na různých linkách

dochází k fragmentaci paketů a může se stát, že se korektně nepřenesou všechny. Proto proces odesílání a příjmu dat probíhá v cyklu, kde se mimojiné kontroluje, zda protistrana skutečně dostala všechna data, která požadovala.

6.4 KIVFS.Net.Secure

Jedná se o rozšíření předchozího modulu, tedy KIVFS.Net, o šifrovaný přenos pomocí protokolu OpenSSL (viz. kapitola č. 5.1.1). Tím, že úzce spolupracuje s předchozím modulem, obsahuje pouze definici externích OpenSSL knihoven a funkci pro inicializaci SSL handshake, které se volá při připojení k serveru:

```
/* Create SSL connection */
/* cipher - KIVFS use RC4-MD5 */
KIVFSNETSECURE SSL* kivfs_net_secure_ssl_handshake(SOCKET sock, char* cipher);
```

a také samozřejmě funkci `disconnect()`. Funkce pro odesílání a příjem dat neobsahuje, neboť je toto již obsaženo ve funkcích `kivfs_net_recv_data()` a `kivfs_net_send_data()`. Řešení je jednoduché, neboť prostým porovnáním na úspěšné zavolání funkce `handshake()` se namísto obyčejného socketu využívá pro komunikaci právě SSL socket.

6.4.1 Instalační balíček pro OpenSSL

Pro provozování OpenSSL šifrování je nutné mít nainstalovaný balíček Win32 OpenSSL Light. Je dostupný ke stažení na stránkách `http://slproweb.com/products/Win32OpenSSL.html`. Projekt Visual Studia je nastaven tak, že hlavičkové soubory jsou umístěny v adresáři *C:\OpenSSL\include* a knihovny v adresáři *C:\OpenSSL\lib*.

6.5 KIVFS.Packing

V tomto modulu se setkáváme s již dříve uvedeným ”zabalováním” a ”rozbalováním” datových paketů. Těmito termíny myslíme proces, kdy do prázdného bloku, předem stanovené velikosti, vkládáme (nebo čteme) na určité indexy buď číselné hodnoty nebo řetězce.

Každý blok může být samozřejmě různě dlouhý a obsahovat různou kombinaci hodnot. Toto je závislé na dané operaci, která se má vykonat a obě strany předem danou strukturu znají.

V rámci tohoto modulu v zásadě používáme hlavně funkce pro kódování/dekódování číselných hodnot a řetězců, proto pro jednoduchost uvedu pouze tyto základní:

```
/* Pack number to data block */
KIVFSPACKING unsigned __int32 kivfs_packing_pack_number(void *data,
    unsigned __int32 index, unsigned __int64 object, int object_size);

/* Pack string to data block */
KIVFSPACKING unsigned __int32 kivfs_packing_pack_string(void *data,
    unsigned __int32 index, char *object);

/* Unpack number from data block */
KIVFSPACKING unsigned __int32 kivfs_packing_unpack_number(void *data,
    unsigned __int32 index, void *object, int object_size);

/* Unpack string from data block */
KIVFSPACKING unsigned __int32 kivfs_packing_unpack_string(void *data,
    unsigned __int32 index, char *object, int object_size);
```

6.6 KIVFS.Common

Tento modul implementuje funkce, které přímo využívají klientské aplikace s podporou KIVFS. Obsahují tedy všechny potřebné funkce z předcházejích modulů od vytváření hlavičky a "zabalování" dat po odeslání či příjem výsledků. Vývojář nepotřebuje znát protokol "do hloubky", ale použije pouze funkce pro konkrétní účel (např. vytvoření adresáře).

Jelikož je si většina funkcí svojí definicí podobná, uvedu na ukázku pouze dvě - žádost o vytvoření adresáře na serveru a žádost o seznam souborů v adresářové struktuře:

```
/* Create remote directory */
KIVFSCOMMON int kivfs_common_mkdir(char *path);
```

```
/* Read actual directory entries and save to structure */
/* my_file - file metadata structure */
KIVFSCOMMON int kivfs_common_readdir(char *path, struct kivfs_core_filedata **my_file);
```

Z výše uvedených popisů funkcí je patrné, že implementovat základní funkcionalitu, jako je např. vytvoření adresáře je velice jednoduché.

6.7 KIVFS.Auth

Podobně jako modul s podporou OpenSSL, řeší tento pouze jednu funkčnost - ověřování uživatelského jména pro přihlášení do aplikace. Na něm jsou pak závislá např. oprávnění k souborům. Autentikační proces je zajištěn technologií Kerberos (viz kapitola 5.1.2).

K ověření uživatelského jména slouží následující funkce, která po zadání potřebných vstupů vrátí úspěch či chybu. Chyba může být způsobena buď neplatným uživatelským jménem a heslem nebo vyhodnocením, že uživatel nemá k serveru přístup.

```
/* Autenticate user via Kerberos */
/* sname - Kerberos service name*/
/* krb_ptype - type of hostname, 3(DNS - use KIVFS),0(none) */
/* krb_ap_req_options - request options, 0 use KIVFS */
KIVFSAUTH int kivfs_auth_login (char *username, char *password, char *hostname,
    char *sname, int krb_ptype, int krb_ap_req_options, int sockfd);
```

6.7.1 Instalační balíček pro Kerberos

Pro podporu autentizace je nutné mít nainstalovaný balíček MIT Kerberos *kfw-3-x-x.exe*. Ten je dostupný ke stažení na stránkách `http://web.mit.edu/Kerberos/dist/`. Dále je pak nutné mít umístěný soubor *krb5.ini* (viz ukázka v příloze A.4) v adresáři *C:\Windows* s korektně uvedenými hodnotami v sekcích `[realms]` a `[domain_realms]`.

6.8 KIVFS.Cache

Posledním modulem knihoven KIVFSTechnology je podpora pro cache. Je to modul volitelný, tudíž pokud z nějakého důvodu není pro implementaci klientské aplikace nutná přítomnost cache, lze jej ze svého návrhu vypustit. Stejně tak například modul autentikační. Modul obsahuje všechny vyjmenované algoritmy pro odstraňování souborů (zmiňované v kapitole 3), logiku pro správu databáze, která uchovávání metadata o každém uchovávaném souboru a počáteční podporu offline režimu.

Pro uchování lokálních metadat každého souboru, ke kterýmu se při práci s cache přistupuje, byl v prvopočátku použit vlastní souborový formát. Tento soubor uchovával nejdůležitější informace o souborech v cache. Dalším vývojem bylo potřeba ukládat kromě většiny metadat souboru také např. pomocné hodnoty a výpočty jednotlivých cache algoritmů. Tento stav začal být velmi neefektivní a náročný. Vlastní souborový formát byl nahrazen za databázové řešení viz kapitola 6.8.1.

Na ukázku jsem vybral dvě funkce. První aktualizuje většinu metadat souboru při přístupu do cache. Druhá demostruje použití vybraného algoritmu pro odstraňování souborů z cache při dosažení hraniční velikosti úložiště.

```
/* Update file metadata in DB */
/* cachename - real filename on file system */
KIVFSCACHE unsigned __int32 kivfs_cache_update_info(
    struct kivfs_core_filedata *core_filedata, char *directory, char *cachename);

/* Classic LRU method for deleting files from cache */
KIVFSCACHE unsigned __int32 kivfs_cache_method_classic_lru(char *cachedir,
    __int32 cachesize, __int64 filesize);
```

6.8.1 Databáze SQLite3

SQLite3[27] je malá knihovna v jazyce C, kterou stačí jednoduše přilinkovat k aplikaci a snadno začít používat. Není tedy založen na principu klient-server a lze tedy tuto databázi snadno implementovat do stávajícího řešení[28]. V SQLite je implementován téměř

celý standard SQL-92 a je tedy možné pomocí obecně známých SQL dotazů přistupovat požadovaná data.

Databáze SQLite3 byla vybrána pro svoji jednoduchost, nenáročnost a snadnou integraci se stávajícím řešením. Je využita právě k uchovávání informací o souborech, které jsou uloženy v cache. Navržená struktura databázové tabulky, uchovávající tyto údaje, je uvedena v příloze (tabulka č. 12). Pro prvotní inicializaci cache není nutné spouštět žádný databázový script, neboť vytvoření této tabulky je začleněno přímo v kódu.

6.8.2 Instalační balíček pro SQLite3

Pro podporu databáze SQLite3 je nutné mít stažený balíček sqlite-dll-win32 a sqlite-amalgation. Je dostupný ke stažení na stránkách `http://www.sqlite.org`. Pro dokončení prostředí je potřeba provést následující postup:

- Utilitou *Visual Studio Command Prompt (2010)*[14] se přesuneme do adresáře s rozbaleným archivem sqlite-dll a spustíme příkaz `LIB /DEF:sqlite3.def`. Tím dojde k vytvoření souboru *sqlite3.lib*, který umístíme do adresáře *C:\sqlite3*
- Rozbalíme archiv sql-amalgation a zkopírujeme soubor *sqlite3.h* taktéž do adresáře *C:\sqlite3*

[14]standardní součást instalace Visual Studio, i Express verze

7 Total Commander

7.1 Implementace

Pro vlastní programování zásuvného modulu jsou dostupné hlavičkové soubory pro jednotlivé typy zásuvných modulů včetně dokumentace. Zásuvné moduly je možné programovat buď v jazyce C/C++ či Borland Delphi. Jedná se o vývoj *DLL knihovny*, jejíž koncovka se přejmenuje podle typu zásuvného modulu, který se vytváří (v našem případě se jedná o WFX[15]).

Zásuvný modul je, stejně jako knihovna KIVFSTechnology, vyvíjen v programovacím jazyce C/C++, tudíž je použit hlavičkový soubor *fsplugin.h*[29] v poslední verzi, tj. 2.1 z července roku 2010. Obsahuje konstanty, struktury a hlavičky jednotlivých funkcí, jejichž tělo doplní programátor o svůj produkční kód.

7.2 Povinné implementace zásuvného modulu

Pro minimální běh zásuvného modulu jsou povinné 4 funkce:

- **FsInit** - inicializace modulu. Uchovává základní hodnoty pro další použití (pořadí zásuvného modulu apod.)
- **FsFindFirst** - načtení prvního souboru/adresáře v adresářové struktuře
- **FsFindNext** - načtení dalšího souboru/adresáře v adresářové struktuře
- **FsFindClose** - ukončení volání předchozích dvou funkcí

Tyto 4 základní funkce realizují načtení a procházení celé adresářové struktury.

7.3 Volitelné implementace zásuvného modulu

Pro úplnou implementaci je nutné přidat ještě další funkcionalitu, tj. rozšířit stávající stav o přenos, vytváření, mazání souborů apod.

[15] zásuvný modul umožňující zpřístupnění lokálních nebo síťových svazků

- **FsMkDir** - vytvoření adresáře
- **FsRemoveDir** - smazání adresáře
- **FsDeleteFile** - smazání souboru
- **FsGetFile** - přenos souboru ze serveru na klienta (download)
- **FsPutFile** - přenos souboru z klienta na server (upload)
- **FsExecuteFile** - spuštění souboru (soubor se přenese do dočasného umístění u klienta a poté spustí)

Všechny tyto funkce lze samozřejmě použít pro více souborů/adresářů. Total Commander pak postupně volá jednotlivé funkce pro splnění požadovaného úkonu. Např. pro přenos adresářové soubory provede *vylistování adresáře -> výběr prvního souboru -> přenos na server -> výběr druhého souboru* apod.

7.4 Další možnosti zásuvného modulu

Pro interakci s uživatelem je možné použít callback funkce `ProgressProc` či `RequestProc`.

Funkce `ProgressProc` umožňuje zobrazení ukazatele stavu přenosu souboru (progressbar), který informuje uživatele o aktuální rychlosti, odhadované době přenosu a o zbývajících procentech do jeho úspěšného ukončení.

Funkce `RequestProc` zobrazuje formulář umožňující vstup od uživatele. Nejčastěji jde o informační okénko s otázkou a potvrzovacími tlačítky `ANO/NE` nebo o vstupní pole vyzívající uživatele k vyplnění např. uživatelského jména či hesla.

7.5 Ukázka implementace

Pro ukázku, jak lze za pomocí knihoven KIVFSTechnology relativně snadno implementovat funkčnost KIVFS, vybral jsem jednu ze základních funkcí, tedy vytvoření vzdáleného adresáře a její kompletní zápis:

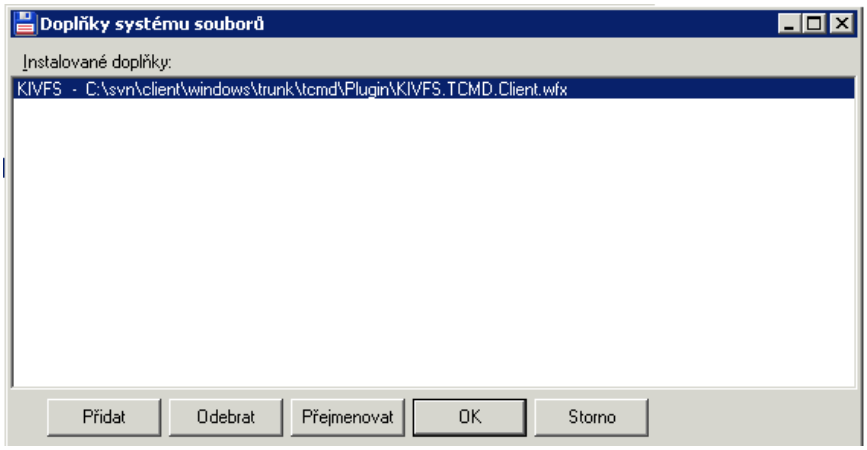
```
BOOL __stdcall FsMkDir(char *Path)
{
    const kivfs_error_t *error = NULL;
    char debugmessage[200];

    int status = kivfs_common_mkdir(Path);

    /* Check for errors */
    if (status < 0)
    {
        error = kivfs_error(status);
        sprintf(debugmessage,"[Create directory] %s \n[%d]: %s",
          Path, error->code, error->description);
        kivfs_message("ERROR", debugmessage);

        return false;
    }

    return true;
}
```

7.6 Nasazení

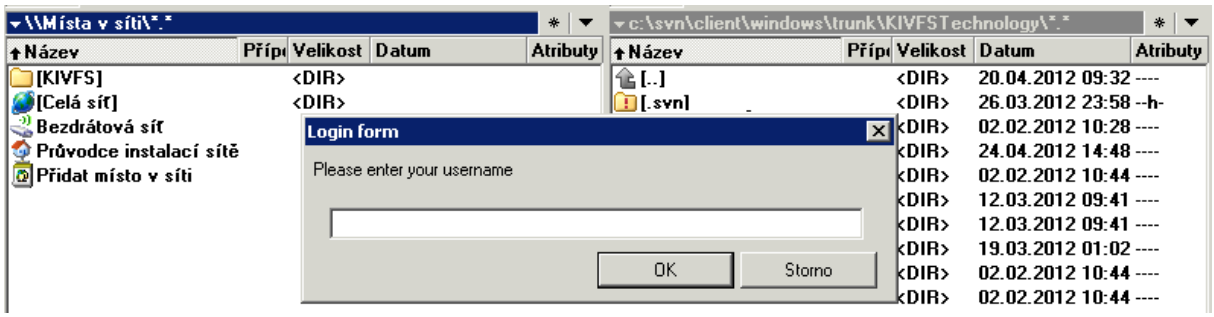
Jelikož je veškerá funcionalita obsažena v knihovnách KIVFSTechnology, je nutné DLL soubory umístit do stejného adresáře, kde je uložen i zásuvný modul. V případě použití cache ještě navíc soubor *sqlite3.dll*.

Instalace modulu se provadí v menu *Konfigurace -> Možnosti -> Doplňky* (viz obrázek č.8).

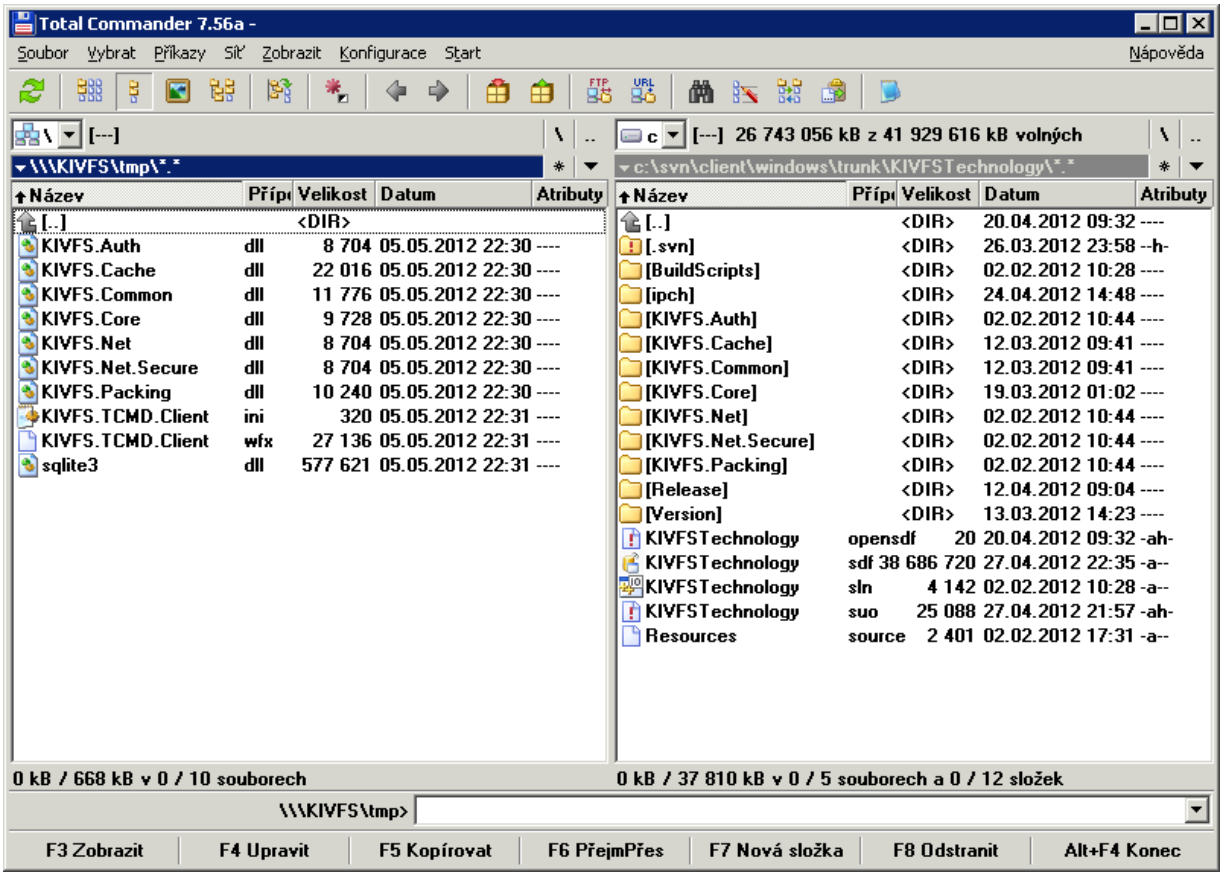
Po zadání cesty k modulu a úspěšném přidání se objeví jako nová položka v nabídce *Místa v síti* a je možné se připojit k serveru. Nejdříve se objeví výzva k zadání hesla (viz obrázek č.9) a po úspěšném přihlášení se uživateli objeví souborová struktura (viz obrázek č.10).



Obrázek 8: Total Commander ukázka - nasazení zásuvného modulu



Obrázek 9: Total Commander ukázka - zadání uživatelského jména



Obrázek 10: Total Commander ukázka - procházení adresářové struktury

7.7 Konfigurace

Zásuvný modul je možné konfigurovat pomocí *.ini souboru*, který se musí nacházet ve stejném adresáři jako samotný plugin. Jeho struktura je stejná, jako většina konfiguračních souborů známých z OS Windows.

7.7.1 Syntaxe konfiguračního souboru

Konfigurace serveru (1 a více)

- **servername** - IP adresa či DNS název serveru
- **serverport** - port služby
- **crypt** - zapnutí/vypnutí šifrovacího mechanismu

- **prior** - priorita serveru [momentálně nepoužíváno]
- **fileblock** - velikost bloku při přenosu souboru

Kerberos autentikace

- **sname** - jméno Kerberos služby
- **hostname** - IP adresa Kerberos serveru

Cache

- **cache** - zapnutí/vypnutí podpory cache
- **cachedir** - adresář, kam se ukládají soubory
- **cachesize** - velikost adresáře v MB
- **cachemethod** - použitá metoda pro odstraňování souborů z cache
- **lru_koef** - koeficient LRU pro zvýhodnění algoritmu LRU&LFU
- **lfu_koef** - koeficient LFU pro zvýhodnění algoritmu LRU&LFU

Volitelné položky

- **debug** - zapnutí/vypnutí logování do souboru

7.7.2 Příklad konfiguračního souboru

Konkrétní příklad funkčního konfiguračního souboru je uveden v příloze A.5.

7.8 Offline režim

Při funkční implementaci cache v klientské aplikaci je dalším krokem podpora offline režimu. Tím je myšlen přístup k souborům i v případě, kdy není připojení k serveru dostupné. V konfiguračním souboru klienta je možné uvést více serverů.

Pokud není ani jeden server online, klientská aplikace se přepne do režimu offline. Tzn. že uživateli začne nabízet soubory, které má uložené v cache. Tedy takové soubory, které již v minulosti stahoval.

V tuto chvíli je podpora offline režimu omezena pouze v režimu `Read-Only`. Je tedy možné procházet adresářovou strukturou, číst metadata souborů, soubory otevírat a kopírovat.

Implementace stávající podpory offline režimu v klienstké aplikaci se skládá ze dvou částí - procházení adresářové struktury a přístup k souborům.

7.8.1 Procházení adresářové struktury

Pokud není ani jeden zadaný servery dostupný, použije se místo funkce

```
kivfs_common_readdir(char *path, struct kivfs_core_filedata **my_file);
```

její offline ekvivalent

```
kivfs_cache_offline_readdir(char *path, struct kivfs_core_filedata **my_file);
```

7.8.2 Přístup k souborům

Offline režim je momentálně realizován pouze v režimu `Read-Only`, tzn. že umožníme uživateli soubory pouze stahovat (poskytujeme je z cache).

Po kontrole, že servery nejsou dostupné a že pracujeme v offline režimu je nutné z cache databáze zjistit název fyzicky uloženého souboru (pole `cachename`). Při uložení souboru do cache úložiště dochází k vygenerování náhodného názvu souboru, aby se předešlo konfliktu názvů. Velmi často se stejný název souboru nachází v různých adresářích na serveru.

Zjištění názvu fyzicky uloženého souboru v cache realizuje funkce

```
kivfs_cache_get_real_cache_name(filename, directory, cachename);
```

Soubor je již poté možné přenést do uživatelem zvoleného adresáře, např. funkcí

```
BOOL WINAPI CopyFile(
  _In_  LPCTSTR lpExistingFileName,
  _In_  LPCTSTR lpNewFileName,
  _In_  BOOL bFailIfExists
);
```

8 Závěr

Hlavním cílem této práce bylo shrnutí vlastnosti dostupných souborových manažerů a jejich využití jako KIVFS klienta. Dále pak ověření možností cache a výkonnost algoritmů pro odstraňování souborů při hraniční velikosti cache úložiště.

Jako vhodný souborový manažer byl vybrán Total Commander pro své největší rozšíření mezi uživateli, výbornou dokumentaci a splňoval všechny požadavky pro kompletní podporu KIVFS. Podpora KIVFS byla úspěšně implementována, včetně šifrování, cache a offline režimu, který je momentálně provozován v módu `Read-Only`. Řešení bylo otestováno na MS Windows XP, 2008 a 7 s Total Commander verze 7.56a.

Nad rámec zadání byl vyvinut soubor knihoven KIVFSTechnology, který v sobě zahrnuje kompletní podporu KIVFS a umožňuje tak snažší vývoj budoucích klientů na operačním systému MS Windows.

Na práci je možné v budoucnu navázat podporou zápisu v offline režimu, další optimalizací cache nebo komprimací přenosů.

Literatura

[1] *Tanenbaum, A., Steen, M.:* **Distributed systems: Principles and paradigms**
Prentice Hall, 2002

[2] *Wikipedia:* **Cache algorithms**
`http://en.wikipedia.org/wiki/Cache_algorithms`

[3] *Groff, J., Weinberg, P.:* **SQL: The Complete Reference, Second Edition**
McGraw-Hill Osborne Media, 2002

[4] *Bžoch, P.:* **Definice cache metod LRU,LFU s využitím metadat ze serveru**
Katedra informatiky a výpočetní techniky ZČU v Plzni

[5] *Žemlička, M.:* **Total Commander - kompletní uživatelská příručka**
Computer press, 2005

[6] *Stross, R.:* **The Microsoft Way: The Real Story Of How The Company Outsmarts Its Competition**
Addison-Wesley, 1997

[7] *Rogers, R., Lombardo, J., Mednieks, Z., Meike, G.:* **Android Application Development: Programming with the Google SDK**
O'Reilly Media, 2009

[8] **Altap Salamander**
`http://www.altap.cz/cz/`

[9] **Speed Commander**
`http://www.speedproject.de/enu/index.html`

[10] **EF Commander**
`http://www.efsoftware.com/cw/e.htm`

[11] **EF Startup**

`http://www.efsoftware.com/su/e.htm`

[12] **EF Mailbox Manager**

`http://www.efsoftware.com/mm/e.htm`

[13] **IBM OS/2**

`http://cs.wikipedia.org/wiki/OS/2`

[14] *Peacock, C.***Windows CE Made Simple**

Butterworth-Heinemann

[15] **Free Commander**

`http://www.freecommander.com/`

[16] **Unreal Commander**

`http://x-diesel.com/`

[17] **Double Commander**

`http://doublecmd.sourceforge.net/`

[18] **Fine Commander homepage**

`http://http://finecmd.kiv.zcu.cz/`

[19] *Cormen, T., Leiserson Ch., Rivest R., Stein C.:* **Introducion to algoritms**

MIT Press and McGraw-Hill, 2009

[20] *Junák, M., Matějka, L., Pešička, L., Pivnička, M., Skupa, J., Strejc, R., Steiner, V., Šafařík, J.:* **KIV-DFS - Experimental Distributed File System**

In Informatics 2009. Košice: Technical University, 2009. s. 45-50. ISBN: 978-80-8086-126-1

[21] *Strejc, R.:* **KIVFS: Souborový a databázový server**

Plzeň, 2009. Bakalářská práce. Západočeská univerzita, Fakulta aplikovaných věd.

[22] *Viega, J., Messier M., Chandra P.:* **Network Security with OpenSSL**
O'Reilly Media, 2002

[23] *Pivnička, M.:* **KIVFS: Zabezpečení, šifrování a ověření identity**
Plzeň, 2009. Bakalářská práce. Západočeská univerzita, Fakulta aplikovaných věd.

[24] *Garman, J.:* **Kerberos: The Definitive Guide**
O'Reilly Media, 2002

[25] *Herout, P.:* **Učebnice jazyka C, 1. a 2. díl**
Kopp, 2002

[26] *Anderson, Ch., Gardner, N., Minutillo, M., Randolph, N.:* **Professional Visual Studio 2010**
Wrox, 2010

[27] *Owens, M.:* **The Definitive Guide to SQLite**
Apress, 2006

[28] *SQLite3*: **C/C++ Interface for SQLite Version 3**
`http://www.sqlite.org/capi3ref.html`

[29] *Ghisler, Ch.:* **Total Commander FS-Plugin writer's guide**
`http://www.totalcmd.net/plugring/fsplugin_guide.html`

A Přílohy

A.1 Měření cache algoritmů

Náhodný přístup						
Cache/HIT	**8 MB**	**16 MB**	**32 MB**	**64 MB**	**128 MB**	**256 MB**
FiFO	53	137	280	560	1126	2278
Random	57	115	292	542	1156	2262
LRU	61	124	293	586	1055	2210
LFU	94	128	294	677	1190	2214
LFU hits	75	166	305	652	1262	2410
LRULFU 1:1	51	148	234	557	1162	2338
LRULFU 1:2	69	159	285	600	1121	2258
LRULFU 2:1	64	120	247	592	1121	2305

Tabulka 7: Měření efektivity cache algoritmů - počet HITů - náhodný přístup

Prioritní přístup						
Cache/HIT	**8 MB**	**16 MB**	**32 MB**	**64 MB**	**128 MB**	**256 MB**
FiFO	266	549	1018	1534	2544	3969
Random	298	568	1036	1603	2546	4039
LRU	279	636	1084	1930	2894	4067
LFU	545	1644	2175	2411	3038	4114
LFU hits	655	1305	2168	2514	3147	4247
LRULFU 1:1	283	728	1220	2207	3058	4191
LRULFU 1:2	415	903	1400	2316	2980	4150
LRULFU 2:1	316	734	1402	2289	3044	4071

Tabulka 8: Měření efektivity cache algoritmů - počet HITů - prioritní přístup

Náhodný přístup						
Cache/[%]	**8 MB**	**16 MB**	**32 MB**	**64 MB**	**128 MB**	**256 MB**
FiFO	0,45	1,33	2,95	5,70	12,55	29,15
Random	0,53	1,13	3,00	5,79	12,87	29,12
LRU	0,59	1,22	2,85	6,14	11,70	28,33
LFU	0,59	1,29	2,71	6,00	13,36	30,31
LFU hits	0,65	1,27	2,83	6,23	12,28	30,08
LRULFU 1:1	0,42	1,38	2,51	5,93	13,46	30,47
LRULFU 1:2	0,61	1,52	2,77	6,54	12,55	29,32
LRULFU 2:1	0,61	1,18	2,55	6,15	12,33	29,65

Tabulka 9: Měření efektivity cache algoritmů - ušetřené přenosy - náhodný přístup

Prioritní přístup						
Cache/[%]	**8 MB**	**16 MB**	**32 MB**	**64 MB**	**128 MB**	**256 MB**
FiFO	2,43	5,41	10,82	17,00	29,38	59,45
Random	2,70	5,60	10,49	17,55	30,13	67,32
LRU	2,77	6,67	12,19	23,03	33,86	59,64
LFU	5,40	21,85	20,45	29,89	42,37	68,98
LFU hits	7,76	17,24	22,80	26,67	48,97	78,31
LRULFU 1:1	3,63	10,95	18,74	33,24	51,36	81,16
LRULFU 1:2	3,45	7,75	13,68	34,77	39,39	80,30
LRULFU 2:1	3,00	11,40	22,25	34,57	50,36	65,22

Tabulka 10: Měření efektivity cache algoritmů - ušetřené přenosy - prioritní přístup

A.2 Nejpoužívanější KIVFS příkazy

název konstanty	**popis**
KIVFS_VFS_READDIR	vypsání obsahu adresáře
KIVFS_VFS_OPEN	otevření souboru
KIVFS_VFS_CLOSE	zavření souboru
KIVFS_VFS_FILE_INFO	získání metadat o souboru
KIVFS_VFS_UNLINK	smazání souboru
KIVFS_VFS_MKDIR	vytvoření adresáře
KIVFS_VFS_RMDIR	smazání adresáře
KIVFS_VFS_READ	stažení soboru
KIVFS_VFS_WRITE	nahrání souboru
KIVFS_FILE_MODE_READ	mód pro stažení souboru
KIVFS_FILE_MODE_READ_WRITE	mód pro přepis souboru
KIVFS_FILE_MODE_WRITE	mód pro nahrání souboru
KIVFS_GLOBAL_HITS	přečtení globálních hitů

Tabulka 11: Nejpoužívanější KIVFS příkazy

A.3 Struktura tabulky SQLite3

název pole	**typ**	**popis**
id	INTEGER PK	identifikátor
directory	STRING	absolutní cesta k souboru
name	STRING	název souboru
cachename	STRING	název souboru na filesystému
version	INTEGER	verze souboru
size	LONG	velikost souboru
mtime	LONG	čas změny souboru
read_hits	INTEGER	počet read hitů
write_hits	INTEGER	počet write hitů
client_hits	FLOAT	nástřel počtu hitů pro LFU algoritmus
local_hits	INTEGER	počet lokálních hitů
plru	FLOAT	priorita LRU pro LRU&LFU algoritmus
plfu	FLOAT	prorita LFU pro LRU&LFU algoritmus
lrulfu	LONG	výsledek LRU&LFU algoritmu

Tabulka 12: SQLite3 struktura cache tabulky FILES

A.4 Ukázka souboru krb5.ini

```
[libdefaults]
default_realm = DFS.ZCU.CZ

[realms]
DFS.ZCU.CZ = {
kdc = fs-1.kiv.zcu.cz
admin_server = fs-1.kiv.zcu.cz
}

[domain_realm]
.kiv.zcu.cz = DFS.ZCU.CZ
kiv.zcu.cz = DFS.ZCU.CZ
```

A.5 Ukázka konfiguračního souboru klienta

```
[server1]
servername=147.228.67.121
serverport=30000
crypt=no
prior=1
fileblock=8192

[server2]
servername=147.228.67.122
serverport=30000
crypt=no
prior=1
fileblock=8192

[server3]
servername=147.228.67.123
serverport=30000
crypt=no
prior=1
fileblock=8192

[auth]
sname=kivfs
hostname=fs-1.kiv.zcu.cz

[cache]
cache=yes
cachedir=c:\temp\kivfscache
cachesize=8
cachemethod=lfu

[other]
debug=yes
```