Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

KIVFS - Webový klient

Plzeň 2013 Karel Hovorka

Prohlášení

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

V Plzni dne 19. května 2013

Karel Hovorka

Abstract

Main goal of this thesis is to create web client for KivFS, distributed file system. Client is implemented in HTML5 with server side proxy in Java. Communication between HTML5 client and server side proxy is performed by new protocol based on HTTP.

Obsah

1 Úvod

V dnešním světě jsou data a informace čím dál důležitější a zařízení různorodější. Potřeba získat data kdekoliv a kdykoliv roste. Data použitá na PC jsou nutná i v notebooku, telefonu nebo jiném zařízení. Jeden z nabízejících způsobů řešení maximální dostupnosti je centrální úložný systém, který bude data spravovat a poskytovat. Nevýhodou takového systému je nemožnost používání v případě selhání systému. Jako alternativní řešení se používá distribuovaný systém, kde při selhání části prvků distribuovaného systému dostupnost není omezena. Aby se data na distribuovaném systému mohla používat, je nutné mít možnost k nim přistupovat - musí existovat klienti. Klient k takovému systému musí být ideálně pro každou možnou platformu. Následující seznam obsahuje jen útržek z používaných platforem, se kterými se může běžný uživatel setkat.

PC, notebooky, netbooky:

- Windows (XP, Vista, 7, 8) [Carpenter(2012)]
- Mac (OS, OS X)[Pogue(2012)]
- Unix/Linux[Evi Nemeth(2010)]
- Solaris[Watters(2005)]
- BSD[Christopher Negus(2008)]

Pro mobilní zařízení, tablety a podobné:

- Windows (Mobile, Phone 7, Phone 8)
- Android[Haseman(2008)]
- iOS[Sadun(2012)]
- Symbian [Sales(2005)]
- Blackberry [Foust(2010)]

Rozdíly mezi platformami jsou často obrovské a je náročné vytvořit pro všechny platformy klienta, protože pro většinu platforem se musí udělat vlastní specifický klient, splňující požadavky systému a používající jeho API.

Jednu věc mají všechny zmíněné platformy společnou - podporují web. Pro každou ze zmíněných platforem existuje internetový prohlížeč. Proto se prostředí www nabízí jako univerzální řešení pro většinu platforem. Prostředí www běží bez nutnosti instalace dalšího softwaru, kromě samotného internetového prohlížeče, který je navíc často už předinstalovaný. Nativní (tzn. vytvořeni speciálně pro platformu) musí pro danou platformu existovat. Nativní klienti zpravidla potřebují instalaci (někdy s potřebou administrátorských práv), nebo musí existovat klient bez nutnosti instalace. Na některých platformách nebude možnost klienta nainstalovat (půjčené zařízení, nedostatek práv). Na druhou stranu nativní klienti nabízí větší komfort a integraci s konkrétní platformou, jako například:

- připojení systému jako diskové jednotky
- jednotný vzhled se zbytkem platformy
- vyšší výkon
- ovládání přirozenější pro platformu
- jednotné klávesové zkratky se zbytkem platformy
- platformě specifické prvky

Pro každodenní používání je jistě vhodnější nativní klient (pokud existuje pro danou platformu). Na cestách, místech kde není možné klienta nainstalovat, nebo platformách s neexistujícím nativním klientem, má webový klient své uplatnění.

2 Analýza

Nejdříve je potřeba si ujasnit, co je distribuovaný souborový systém, co je specifické pro systém KivFS a jaké jsou požadavky pro vytvoření webového klienta.

2.1 Distribuovaný souborový systém

Distribuovaný souborový systém je množina nezávislých počítačů, které se jeví jako jeden souborový systém. Uživatel se připojí na jakýkoliv z těchto počítačů a může pracovat se svými soubory. Při dalším připojení k jakémukoliv počítači distribuovaného souborového systému jsou soubory ve stavu jako při posledním připojení.

Distribuovaný souborový systém, jako každý distribuovaný systém, musí splňovat několik základních požadavků:

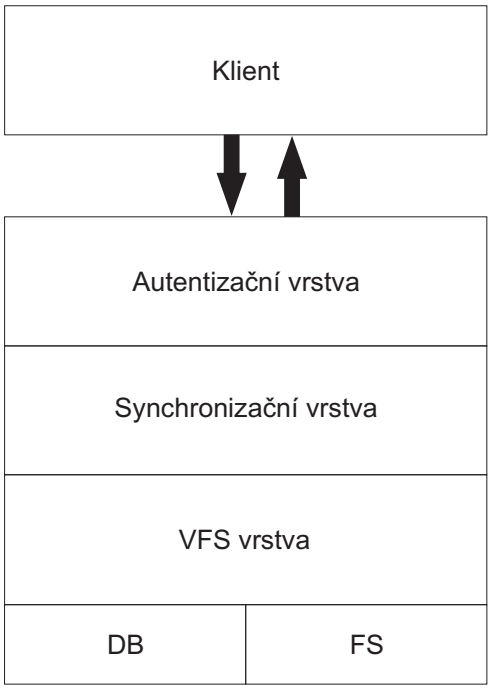
- Transparentnost přístupu
  Systém se musí tvářit jako jeden zdroj, přestože fyzicky je distribuovaný systém sestavený z více systémů.
- Škálovatelnost
  Výkon systému musí být možné přizpůsobit požadavkům. Navýšení výkonu se realizuje přidáním dalšího serveru do systému.
- Konzistentnost
  Všechny operace musí proběhnout atomicky. Při změně souboru na serveru se musí objevit změny i na všech ostatních serverech (replikace). Pokud se operace nedokončí, všechny servery musí skončit ve stavu před začátkem operace.
- Bezpečnost
  Server musí být dobře zabezpečený na úrovni přístupu k samotnému serveru, přístupu k datům a operací nad daty.
- Stabilita
  Distribuovaný systém je funkční i v případě výpadků. V případě selhání jednoho nebo několika serverů systém pořád funguje.

Některé existující distribuované souborové systémy:

- AFS [Campbell(1998)]
- NFS 4[Callaghan(2000)]
- Coda [Braam(2012)]
- KivFS

2.1.1 KivFS

KivFS je distribuovaný souborový systém vyvíjený na katedře informatiky Západočeské Univerzity v Plzni. Cílem systému je odstranit nedostatky existujících distribuovaných souborových systémů a vytvořit souborový systém jako náhradu za distribuovaný souborový systém AFS.



Obrázek 2.1: Vrstvy KivFS.

Server

Server je rozdělený na několik vrstev, které dohromady poskytují služby souborového systému:

1. Autentizační vrstva
   Tato vrstva se stará o bezpečnost - autentizuje uživatele a naváže šifrované spojení. Spojení je šifrované pomocí SSL[Rescorla(2000)] a používá se šifrování AES 256. Autentizace probíhá protokolem Kerberos[Tung(1999)]. Pokud se klient autorizuje, autentizační vrstva dále přeposílá příkazy další vrstvě. Vrstva slouží jako proxy a je to jediný vstupní bod do celého systému pro klienty.

2. Synchronizační vrstva
   Synchronizační vrstva synchronizuje příkazy mezi servery, zajišťuje správnou konzistenci dat na serveru a atomičnost operací. Komunikace mezi servery probíhá touto vrstvou.

3. VFS
   VFS je vrstva samotného virtuálního souborového systému, která poskytuje služby pro práci se souborovým systémem a odstraňuje závislosti na konkrétním souborovém systému.

4. Datová a souborová vrstva
   Datová a souborová vrstva jsou 2 komponenty, které fyzicky ukládají soubory a informace o nich. Jedna část ukládá samotné soubory. Soubory jsou uloženy na lokálním souborovém systému. Pro souborové úložiště se používá souborový systém XFS, ale je možné použít i jiný. Druhá část ukládá metadata o souborech (cesta, práva, verze a další), které jsou uloženy v relační databázi. Jako relační databáze se používá MySQL, ale je možné použít i jiné relační databáze.

Vrstvy jsou zobrazeny na obrázku 2.1

Klienti

Aktuálně existují pro KivFS klienti pro následující platformy:

- FUSE [Jaroš(2012)]

- Jádro Linux [Fazekaš(2010)]
- Total commander plugin
- Java [Hovorka(2010)] (Má bakalářská práce)
- Android (Má semestrální práce z KIV/MKZ)

Protokol

Komunikace s KivFS probíhá KivFS protokolem běžícím nad TCP/IP protokolem. Každá zpráva od klienta má hlavičku, kterou může následovat tělo s daty. Každá odpověď od serveru má také hlavičku, kterou může následovat tělo s daty, pokud server vrací data (například při výpisu adresáře, požadavku na detail souboru).

Hlavička každé zprávy má následující strukturu definovanou v jazyce C:

```
typedef struct {
/*
 * Konstantní číslo definované v konfiguraci sloužící
 * pro kontrolu správnosti hlavičky a identifikaci KivFS
 * paketu, v našem případě 31337.
 */
    uint32 magicnumber;
/*
 * Čas příkazu. Číslo vyjadřuje počet milisekund od 1.1.1970
 * UTC.
 */
    uint64 timestamp;
/*
 * Kód příkazu. Například u autentizace je to konstanta
 * KivFS\_PASSWD s~hodnotou 38, v případě odpovědi je
 * o 10000 vyšší.
 */
    uint32 command;
/*
 * Návratový kód. V případě požadavku je nulový, v případě
 * odpovědi je také nulový v případě úspěchu. V případě
 * neúspěchu obsahuje zápornou hodnotu vyjadřující odpověď.
```

```
 */
    int32 return_code;
/*
 * Vyjadřuje velikost dat v bajtech.
 */
    uint64 data_size;
/*
 * Obsahuje id uživatele - používané vrstvami pod autorizační
 * vrstvou.
 */
    uint64 session;
/*
 * Proměnná pro další vrstvy na serveru, nastaveno na 1.
 */
    int32 type;
} kivfs_msg_head_t;
```

magic_number	31337
timestamp	123456789123
command	READ_DIR
return_code	0
data_size	8
session	1
type	1
data	4 /TMP

Tabulka 2.1: Naznačená struktura KivFS zprávy požadavku pro výpis adresáře. Data jsou číslo vyjadřující počet znaků řetězce (cesta) a pak samotný řetězec.

Nejpoužívanější kódy příkazů jsou popsané v následující tabulce:

Nejpoužívanější kódy odpovědí jsou popsané v následující tabulce:

Data jsou buď řetězce obsahující cesty k souborům, nebo informace o souborech. Samotný přenos souborů je rozebraný v následující kapitole.

Pokud je požadován detail souboru nebo seznam adresáře, jednotlivé soubory mají následující strukturu:

```
typedef struct {
```

KIVFS_MKDIR	4	vytvoření adresáře
KIVFS_UNLINK	5	smazání souboru
KIVFS_RMDIR	6	smazání adresáře
KIVFS_MOVE	7	přesun souboru/adresáře
KIVFS_OPEN	39	otevření souboru
KIVFS_CLOSE	40	zavření souboru

Tabulka 2.2: Kódy příkazů v KivFS.

KIVFS_OK	0	žádná chyba
KIVFS_ERROR	-1	obecná chyba
KIVFS_SYN_ERROR	-2	chyba na synchronizační vrstvě
KIVFS_VFS_ERROR	-3	chyba na VFS vrstvě
KIVFS_AUTH_ERROR	-4	chyba na autorizační vrstvě
KIVFS_DB_ERROR	-6	chyba na databázovém serveru
KIVFS_FS_ERROR	-7	chyba na souborovém systému
KIVFS_ERC_SERVER_IS_BUSY	-11	chyba, server je zaneprázdněn
KIVFS_ERC_FILE_EXISTS	-15	chyba, nelze vytvořit existující soubor
KIVFS_ERC_DIR_EXISTS	-16	chyba, nelze vytvořit existující adresář
KIVFS_ERC_NO_SUCH_FILE	-17	chyba, soubor nenalezen
KIVFS_ERC_NO_SUCH_DIR	-18	chyba, adresá nenalezen
KIVFS_ERC_FILE_LOCKED	-24	chyba, soubor je zamčený
KIVFS_ERC_DIR_NOT_EMPTY	-31	chyba, adresář není prázdný
KIVFS_ERC_NO_SPACE_LEFT	-45	chyba, nedostatek místa na úložišti

Tabulka 2.3: Kódy odpovědí v KivFS.

```
/*
 * Enum reprezentující typ souboru (adresář, soubor).
 * Adresář má hodnotu 1, soubor hodnotu 2.
 */
kivfs_file_type_t type;
/*
 * Název souboru.
 */
char *name;
/*
 * Jméno vlastníka souboru.
 */
```

```
char *owner;
/*
 * Jméno skupiny souboru.
 */
char *group;
/*
 * Velikost souboru.
 */
uint64 size;
/*
 * Datum modifikace souboru.
 */
uint64 mtime;
/*
 * Datum přístupu (access) souboru.
 */
uint64 atime;
/*
 * Verze souboru - při každé změně se zvedne o 1.
 */
uint64 version;
/*
 * Hodnota počítající čtení souboru.
 */
uint64 read_hits;
/*
 * Hodnota počítající zápis do souboru.
 */
uint64 write_hits;
/*
 * Obsahuje informace o přístupových právech k souboru
 * pro různé uživatele a různé operace.
 */
kivfs_acl_t *acl;
} kivfs_file_t;
```

Struktura práv je podobná modelu práv v systémech unix [David P. Bovet(2005)]. Je zde nastavení pro zápis/čtení/spuštění (nebo vypsání adresáře) pro uživatele, skupinu a ostatní.

Přenos souborů

Pro práci s daty souboru se napřed musí soubor otevřít příkazem open. Návratovou hodnotou příkazu open je file deskriptor - číslo, které reprezentuje vazbu na otevření souboru. Pro další operace se používá tento file deskriptor. Při operacích se pošle file deskriptor, typ operace (čtení/zápis) a rozsah čtení. Jako odpověď přijde ip adresa a port, na který se musí klient připojit. V tomto připojení dojde k samotnému čtení nebo zápisu, a pak se připojení ukončí.

Pro zakončení práce se souborem se musí poslat příkaz flush a následně příkaz close.

2.2 KivFS a prostředí WWW

Pro vytvoření klienta pro KivFS v prostředí www je potřeba nejprve rozebrat dostupné webové technologie, a pak analyzovat možnosti implementace.

2.3 Webové technologie

Web jako platforma a technologie je jedna z nejpoužívanějších a nejrychleji se rozvíjejících vůbec. Přestože web vznikl před necelými 25 lety, dnešní IT svět je už bez něj těžko představitelný. Používá se k vyhledávání informací, komunikaci, zpravodajství, ...

2.3.1 Dělení

Z hlediska vrstev se webové technologie dělí na 3 části. První je serverová část, která generuje data a posílá je klientům. Serverová část se dělí ještě na www servery (Apache HTTP Server, Nginx, Lighttpd, IIS, servletové kontejnery Tomcat, Jetty, ...) a technologie, které na www serverech běží (PHP[Lurig(2008)], ASP.NET[Bill Evjen(2008)], Java EE[Kevin Mukhar(2006)], Node.js[Mike Cantelon(2013)], Ruby on rails[Hartl(2012)], Google App Engine[Sanderson(2012)] a mnoho dalších). Technologií je mnoho a pořád nové vznikají.

Druhá část je transportní část, která obsahuje protokoly pro přenos dat. V této části jsou protokoly HTTP a WebSocket.

Třetí je klientská část, která zpracovává data a zasílá požadavky na server, obsahuje technologie jako je HTML, Javascript, kaskádové styly CSS a další jako Java Applety[K. C. Hopson(1996)], Adobe Flash[Team(2012)], ActiveX[Armstrong(1997)]. .

2.3.2 HTTP Servery

HTTP server je program, který implementuje protokol HTTP a zpravidla skrz protokol poskytuje soubory nebo služby. HTTP serverů je velké množství. Liší se použitím, kvalitou i cenou. Na internetu jsou nejpoužívanější Apache[Engelschall(2000)], Internet Information Services (IIS)
[Mike Volodarsky(2008)] a Nginx[Nedelcu(2010)], které se používají podle průzkumů na cca 95% webů. Kromě těchto serverů existují ještě další servery se specifickou funkčností, jako třeba proxy.

Apache HTTP Server

Apache je nejrozšířenější HTTP server. Jeho vývoj začal v roce 1993 a už v roce 1996 se stal nejpopulárnějším HTTP serverem na internetu. Podle aktuálních statistik se používá na cca 62% serverů. Apache je vyvíjen s open-source licencí, která umožňuje používat zdrojové kódy v C i binární distribuce zdarma pro většinu platforem včetně Windows, Linux, BSD, Solaris, Mac OS X a dalších. Funkčnost Apache je velmi rozsáhlá pro velké množství modulů.

IIS

Server IIS je vyvíjený firmou Microsoft a veškerá jeho funkčnost je proto velmi silně integrovaná s dalšími produkty firmy Microsoft, jako je ASP.NET nebo Active Directory. Podle aktuálních statistik se používá na cca 17% serverů.

Nginx

Nginx je malý server, který se původně specializoval na základní funkčnost a soustřeďoval na efektivitu. Je vhodný pro odbavování velkého množství statického obsahu nebo PHP skriptů. Podle aktuálních statistik se používá na cca 16% serverů.

Lighttpd

Lighttpd[Bogus(2008)] je oblíbený pro svou nízkou paměťovou náročnost a stabilitu v kritických bodech velkých webových aplikací. Často se používá jako backend.

Java servletové kontejnery

Java servletové kontejnery jsou webové servery, které umí pouštět Java EE webové aplikace. Implementací servletových kontejnerů je velké množství, mezi nejčastěji používané patří kontejnery JBoss[Javid Jamae(2009)], Apache Tomcat[Jason Brittain(2003)], Jetty[Roden(2010)], Glassfish[Kou(2010)], IBM Websphere[Rama Turaga(2006)]. Tyto servery se málokdy používají jako samostatný webserver a často je používán pro vstupní bod nějaký jiný webový server jako proxy, například Apache nebo Lighttpd.

2.3.3 Serverové technologie

PHP

PHP[Lurig(2008)] je momentálně nejrozšířenější serverová webová technologie. Je rozšířená díky své jednoduchosti použití při tvorbě dynamických stránek. Jazyk vychází ze syntaxe jazyka C. Základní knihovní funkce jsou podobné C jako je práce s řetězci, soubory a podobně. Jazyk je interpretovaný a dynamicky typovaný. Nevýhodou jazyka je nemožnost stavovosti - skript je spuštěn při začátku požadavku a po poslání odpovědi končí. Pro udržení informací o stavu se dá používat *session*.

ASP.NET

ASP.NET[Bill Evjen(2008)] je součást platformy .NET, která se soustřeďuje na webové technologie. Webové aplikace v .NETu se mohou psát v několika jazycích (C#, Visual Basic .NET, J#) se stejnou sémantickou hodnotou, a následně se převede do Common Language Runtime. Celou technologii vytváří a silně podporuje firma MicroSoft, a proto je tato technologie úzce spjatá s dalšími produkty od společnosti MicroSoft jako je operační systém Windows a server IIS. Výhodou této technologie je velká jednotnost - jedno kvalitní oficiální vývojové prostředí, návrhové vzory a frameworky. Nevýhodou je horší přenosnost aplikací na jiné platformy - existují jiné implementace (například mono pro linux), ale ty nejsou 100% bezproblémové. Další nevýhodou je cena, protože operační systém Windows je placený.

Java EE

Java EE (Enterprise Edition) [Kevin Mukhar(2006)]je součást platformy Java zaměřená na webové aplikace. Jedná se o nádstavbu nad Javu SE s přidanými knihovnami pro webové aplikace a dalšími takzvanými enterprise technologiemi. Základem technologie jsou Servlety, což jsou třídy, které obsluhují HTTP požadavky. Celá aplikace potom běží v servletovém kontejneru. V jednom kontejneru může zpravidla běžet najednou více aplikací. Aplikace běží od začátku spuštění v servletovém kontejneru. Při HTTP požadavcích se spustí vlákno obslužného serveru, na pozadí ale mohou běžet další vlákna nezávisle na požadavcích. Snadno je proto možné během jednoho požadavku nechat informace uložené v paměti a při dalším požadavku je opět načíst. Takto se snadno dají udržovat například TCP spojení na pozadí. Výhodou této technologie je stejně, jako celé Javy, platformová nezávislost a nulová pořizovací cena.

Nevýhodou je nižší výkonnost, náročnost na operační paměť a velká roztříštěnost frameworků třetích stran.

2.3.4 Protokol HTTP

Nesdílnou součástí webu je protokol HTTP, který byl navržen společně s prvním HTML. Programům, které implementují HTTP protokol a poskytují (mimo jiné) webové stránky, se říká webové nebo také HTTP servery. Proto-

kol funguje nad spojovaným protokolem, TCP na portu 80. Jeho šifrovanou formou je protokol HTTPS, který zpravidla běží na portu 443. Protokol je synchronní (požadavek-odpověď) a po odeslání odpovědi je spojení ukončeno (pokud není řečeno jinak, např. hlavička *Connection: keep-alive*, kdy se může použít 1 spojení pro více požadavků a odpovědí).

Verze protokolu

- HTTP 0.9

  0.9 je originální publikovaná verze, která obsahuje jen jednu metodu - GET. Klient se napojí a v plaintextu napíše na řádku "GET cesta". Server poté pošle odpověď - obsah dokumentu, jako plaintext v *ascii* kódování, a spojení ukončí. Nebylo možné zjistit správnost požadavku jinak než obsahem výsledného dokumentu (žádný status kód).

- HTTP 1.0

  Protokol 1.0 přidal možnost metadat, hlavičky, další metody a přivedl protokol do formy, ve které ho známe dnes. Každý request napřed obsahuje metadata, pak následuje prázdná řádka, a poté data. Objevila se metoda POST, která umí posílat i soubory, MIME typy pro posílání souborů jakéhokoliv typu, i jako odpověď.

- HTTP 1.1

  Verze 1.1 přinesla změny především standardizováním dalších hlaviček, status kódů a metody OPTIONS, která sloužila k získání vlastností serveru. Hlavičky, které jistě stojí za zmínění, jsou:

  - Host
    Hlavička host umožňuje dotazovat se na hostname. Umožní, aby na serveru běželo více domén a touto hlavičkou se vybrala požadovaná doména.
  - Range
    Tato hlavička umožňuje získání jen části obsahu dat a tím ušetření přenosu, pokud chce klient jen část dat. Toto je užitečné například při stahování souboru po částech.

Formát požadavku

Formát zprávy požadavku má následující tvar (HTTP 1.1):

```
GET /index.HTML HTTP/1.1
Connection: close
Accept: text/HTML
```

Požadavek na první řádce obsahuje použitou metodu (například GET, HEAD, POST, PUT, DELETE, OPTIONS), cestu k požadovanému dokumentu a verzi protokolu. Na dalších řádkách jsou metadata - hlavičky protokolu. Ty udávají další informace o požadavku a požadavky na odpověď jako například:

- kódování požadavku/odpovědi (Accept-Charset, Content-Type)
- informace o klientovi (User-Agent)
- akceptovaný typ dokumentu odpovědi (Accept)
- komprese (Accept-Encoding, Content-Encoding)
- autorizační údaje (Authorization, WWW-Authenticate)
- časové údaje (Date)
- nastavení cache (Cache-Control, Pragma)
- cookies (Set-Cookie, Cookie)

Hlaviček je velmi mnoho, někteří klienti používají vlastní nestandardní hlavičky. Pokud nejsou tyto hlavičky serverem rozpoznány, tak jsou ignorovány. Po hlavičkách následuje 1 prázdná řádka. Po prázdné řádce mohou následovat data, pokud klient chce nějaké poslat (například u metody POST se přenáší data zde).

Formát odpovědi

Formát odpovědi má následující tvar (HTTP 1.1):

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Content-Length: 5

hello
```

Na první řádce je napřed verze protokolu, poté status kód, který má stručně číselně popisovat výsledek. Kód je v rozmezí 200-599:

- 1xx - Informační (pokračujte, změna protokolu), u HTTP se používá například při handshake a přechodu na protokol WebSocket.
- 2xx - Úspěch, doplňující informace o úspěchu, u HTTP se používá prakticky jen hodnota 200 znamenající "OK".
- 3xx - Přesměrování, dočasné přesměrování, obsah nezměněn a podobné.
- 4xx - Chyba klienta, požadavek má špatný formát nebo nemůže být splněn. Typická je chyba 404, která se používá v případě, že cesta k dokumentu nebyla na serveru nalezena nebo 403 při neoprávněném přístupu.
- 5xx - Chyba serveru, požadavek se nepodařilo obsloužit z důvodu chyby serveru, typicky chyba 500 při obecné chybě serveru. Chyba většinou znamená, že server je ve stavu, ve kterém není schopný správně obsluhovat a je nutná údržba.

Po číselném kódu je ještě jeho krátký textový popis, který není pevně daný a může se měnit. Přesto se používají u určitých kódů standardní popisy (200-OK, 404 Not Found atd). Směrodatný je číselný kód.

Po první řádce následují řádky s metadaty - hlavičkami protokolu, stejně jako při požadavku. Po hlavičkách je prázdná řádka a po ní ve většině případů data (například metoda HEAD naopak neposílá data - je identická s metodou GET, jen nepošle data). Po odeslání dat se spojení ukončí, pokud není použita hlavička *Connection: keep-alive*.

Cookies

Speciálním mechanismem protokolu pro udržení stavu jsou takzvané *Cookies*. Podstatou *cookie* je identifikace posloupnosti příkazů pro jednoho uživatele. Server při vyžadování používání *cookie* pošle hlavičku *Set-Cookie*. Hodnota *Set-Cookie* obsahuje jméno proměnné, hodnotu proměnné, trvanlivost, doménu a adresář na serveru, pro který *cookie* platí. Prohlížeč od té doby tento *cookie* (proměnnou a hodnotu) posílá s každým požadavkem na daný server (doménu) s nastavenou cookie hlavičkou *Cookie*. Díky tomuto mechanismu

může server rozeznávat uživatele, posloupnosti příkazů, ukládat nastavení a další.

Tento mechanismus je ale také velmi zranitelný. Často stačí poslat požadavek webové aplikace s cookies, které jsou odposlechnuté ze sítě, a vydávat se za uživatele, kterému byly cookies ukradeny. Jiný typ útoku je CSRF (cross-site request forgery) [Pinto(2007)], který využívá *cookies* jiným způsobem. Na jakýkoliv web přidáme obrázek:

```
<img src="www.domenapodutokem.cz/uzivatel/smazat" />
```

Odkaz na obrázek se samozřejmě nezobrazí, prohlížeč ale požadavek pošle včetně cookies a požadavek může takto například smazat uživatele, jak je znázorněno v odkazu na obrázek.

2.3.5 HTML

Vznik

Jazyk HTML vytvořili koncem osmdesátých let minulého století fyzik CERNu Tim Berners-Lee a informatik Robert Caillau. Původním účelem HTML bylo sdílet vědecké dokumenty po celém světě a umožnit přímé odkazy mezi dokumenty (hypertext). Základní jazyk uměl jen několik značek na zvýraznění, strukturování textu a samozřejmě odkazy. V roce 1991 byly specifikace jazyka a ovládací software (jednoduchý prohlížeč, editor) uvolněny (neoficiálně "HTML 1.0"). V roce 1992 se objevují první další prohlížeče. Tyto prohlížeče ještě neměly grafiku a byly pouze textové.

Původní verze jazyka přestala stačit, výrobci prohlížečů si začali vytvářet vlastní značky, které nebyly kompatibilní s dalšími prohlížeči, a proto Berners-Lee začal vytvářet standard HTML 2 pod záštitou IETF (Internet Engineering Task Force), aby zachoval jednotnost jazyka.

HTML 2

Na druhé verzi HTML se začalo pracovat v roce 1993. Hlavním smyslem byl sběr proprietárních značek různých prohlížečů ve snaze tyto značky sjednotit

a vytvořit ucelený standard, kterým se budou řídit všichni. Kromě sjednocení ještě přidává formulářové prvky. Konečná verze vyšla v roce 1995.

HTML 3.0

HTML 3.0 vznikl v roce 1995 jako neoficiální rozšíření od firmy Netscape pro prohlížeč Netscape Navigator, vyvíjený od roku 1994.

HTML 3.2

V roce 1996 se začíná dělat na verzi 3.2, která je opět snahou sjednocení předchozích verzí. Spousty rozšíření z neoficiální verze 3.0 nebyly v 3.2 zahrnuty. Hlavním rozšířením jsou tabulky. Verze 3.2 byla oficiálně dokončena v roce 1997.

HTML 4.0 a 4.01

Souběžně s vydáním 3.2 se začala vyvíjet další generace HTML - 4.0, která obsahovala rámce, Javascript a vkládání obecných objektů (jako Flash a Applet). Verze odstranila velké množství původních tagů, které byly příliš zaměřené na zobrazení a neměly sémantickou hodnotu, například tag *center*.

Specifikace HTML 4 se dělí na 3 větve, které mají různě striktní pravidla.

- strict
  Obsahuje všechny HTML elementy a atributy kromě elementů, které obsahují zobrazovací hodnotu. Rámce nejsou povoleny.
- transitional
  Stejně jako strict, ale obsahující i elementy se zobrazovací hodnotou. Rámce nejsou povoleny.
- frameset
  Stejně jako transitional, ale jsou povoleny rámce.

XHTML

XHTML je větev HTML4. Je to XML forma, která splňuje normy xml jazyka nepovinné pro HTML jako jsou zavřené značky, atributy v uvozovkách, nutnost hodnoty atributů. Kromě xml formy XHTML 1.0 a 1.1 nic nepřidává. S vývojem XHTML se dále pokračovalo s verzí 2.0, která byla ale příliš revoluční a příliš se soustřeďovala na správnost jazyka, což by brzdilo přirozený rozvoj webu a nakonec byla tato větev na webu opuštěna. Verze 1.0 i 1.1 se opět dělí na verze strict, transitional a frameset.

Kompatibilita

Každý dokument je identifikovatelný hlavičkou v dokumentu, prohlížeče pomocí hlavičky dokumentu identifikují starší verzi HTML a interpretují dokument podle verze. Běžně se používá nejvíce HTML4.01, XHTML1.0, XHTML1.1 a HTML5.

HTML5

HTML5 je posledním vývojovým stupněm jazyka HTML, této technologii je věnovaná samostatná sekce 2.4.

2.3.6 Kaskádové styly CSS

Kaskádové styly vznikly jako snaha o konzistentní způsob řízení vzhledu webových stránek, aby samotná struktura a obsah stránky byly nezávislé na vzhledu. Jazyk se skládá ze selektorů, které identifikují elementy HTML stránky a atributů, které se vybraným elementům nastaví. CSS nejprve umožňovalo jen menší úpravy jako je změna barvy, fontu (podobné věci umělo i HTML, ale byla snaha o čistší HTML, aby HTML dokument obsahoval pouze obsah a ne vzhled). Později se přidaly další možnosti jako je pozicování, transformace nebo animace.

CSS 1

Specifikace CSS 1 byla dokončena v roce 1996. Specifikace už obsahovala selektory a pravidla v klasickém formátu. Pravidla se ve většině týkala textu, písma a základního pozicování.

CSS 2.1

CSS 2.1 přidalo další možnosti ohledně pozicování (atribut position) a snížení nutnosti modifikovat obsah kvůli formě. Tato verze byla dokončena ve finální formě v červnu 2011.

CSS 3

CSS 3 dále zvyšuje možnosti pozicování a přidává další pokročilejší způsoby zobrazení jako jsou transformace (i 3d) a animace. Kromě nových atributů přibyla řada selektorů a pseudoselektorů nebo queries, které umožňují pro různé typy zařízení aplikovat různé styly (obrazovka, tiskárna, mobilní zařízení). Specifikace se stále vyvíjí jako součást HTML5.

2.3.7 Javascript

Javascript je interpretovaný jazyk vytvořený firmou Netscape jako interpretovaný jazyk vhodný pro běh v prohlížeči. Netscape chtěl použít jazyk lehčí a jednodušší než například Java, aby bylo pro tvůrce stránek snažší ho používat. Jazyk se původně jmenoval Mocha a na Javascript byl přejmenován až v roce 1995. V roce 1996 byl jazyk standardizován s názvem ECMAScript, což je dodnes oficiální název, přestože se spíš používá Javascript.

Dnes už je Javascript standardním vybavením každého většího prohlížeče a stále se vyvíjí.

Existuje velké množství frameworků a knihoven zaměřených na templatování (dust.js), hry (crafty.js), sjednocení chování pro různé prohlížeče (jquery, prototype) a další.

Jazyk je kritizován pro nedostatky v syntaxi a funkčnosti a proto pro něj

existuje mnoho jazykových nadstaveb, které jsou buď přímo interpretovány prohlížečem nebo kompilovány do Javascriptu jako třeba Coffeescript [MacCaw(2012)]. Některé nadstavby umožňují nové programové konstrukce, které samotný Javascript neumí, jako například rozhraní.

S Javascriptem úzce souvisí způsob zápisu dat *json* (JavaScript Object Notation), který se používá pro strukturovaná data. Jeho výhodou je platformová nezávislost, čitelnost a jednotnost.

2.4 HTML5

HTML5 je soubor moderních webových technologií - Javascriptu, CSS3 a jazyka HTML. Od verze 5 se nepočítá, že budou přibývat další verze - jednou za čas se část specifikace uzavře (potom, co bude ve většině prohlížečů implementovaná beze změn) a bude se pokračovat ve vývoji dále.

2.4.1 Novinky v HTML5

HTML5 navazuje na HTML 4.01 a CSS 2.1.

Nové elementy

HTML5 klade velký důraz na sémantiku - přibylo mnoho nových elementů se sémantickou hodnotou, jako například:

- article
  Blokový element vyjadřující článek.
- section
  Blokový element vyjadřující sekci.
- footer, header
  Blokové elementy vyjadřující patičku a hlavičku.
- aside
  Blokový element vyjadřující sekundární obsah.

- nav
  Blokový element vyjadřující hlavní navigaci webu.
- time
  Inline element vyjadřující časový údaj.

Kromě toho přibylo mnoho dalších funkčních elementů. Některé ještě zdaleka nejsou podporovány nebo nejsou podporovány v plném rozsahu ve všech prohlížečích. Mezi nejzajímavější patří:

- audio, video
  Elementy na přehrávání audia a videa.
- details
  Blokový element, který vyjadřuje informaci, která obsahuje i další doplňující informace, které si může uživatel zobrazit *rozkliknutím* základního popisu (element summary).
- figure
  Blokový element s vlastním obsahem, který definuje například obrázek, tabulku, fotografii.
- progress
  Blokový element reprezentující průběh procesu probíhajícího na stránce.
- command
  Element, který reprezentuje tlačítko s příkazem, které může uživatel vyvolat.
- canvas
  Element, který obsahuje rastrovou grafiku a umožňuje s ní pracovat.
- datalist
  Element, který obsahuje textové informace použitelné pro našeptávání formulářovými prvky.

Formuláře

Formuláře byly velmi vylepšeny z uživatelského hlediska. Formulářové prvky mohou být mimo element form a asociace je pomocí atributu form. Formulářový prvek pro upload souborů umožňuje vybrání více souborů najednou

pomocí atributu *multiple*. Každý formulářový prvek může mít příznak *required*, který zajistí, aby se formulář neodesílal, pokud nebude prvek vyplněný. Přibylo několik typů vstupu elementu input, které usnadňují zadávání informací. Tyto nové typy jsou obzvlášť užitečné v zařízeních, kde zadávání může být náročnější (mobilní telefony). Mezi nové typy patří:

- color
  Formulářový prvek pro zadávání barvy.
- date, datetime, week, month
  Formulářové prvky pro zadávání časových údajů.
- number, range
  Formulářové prvky pro zadávání číselných údajů.
- tel
  Formulářový prvek pro zadávání telefonního čísla.
- url
  Formulářový prvek pro zadávání internetové adresy.
- search
  Formulářový prvek pro vyhledávání.

Textově založené formulářové prvky mají navíc možnost našeptávající nápovědy pomocí elementu *datalist*, který obsahuje informace k našeptávání. K našeptávání stačí nastavit atribut *list* na hodnotu stejnou, jako je hodnota atributu *id* elementu *datalist*.

Javascript

V Javascriptu přibylo velké množství nových API, tady je výčet těch nejzajímavějších:

- Geolocation API
  API pro získávání polohy klienta. Poloha může být získána různými způsoby - například pomocí IP adresy, GPS, GSM nebo Wifi. Aby aplikace mohla získávat polohu, musí mít povolení od uživatele.

- Localstorage
  API pro ukládání dat lokálně, které jsou dostupné při příští návštěvě stránky. Data jsou ve formátu klíč-hodnota.

- Sessionstorage
  API pro ukládání dat lokálně, které jsou dostupné během práce stejně jako u Localstorage, ale data jsou smazány při zavření prohlížeče nebo tabu.

- File API
  API umožňuje práci se soubory, zpřístupňováním a modifikací jejich dat. Je mnoho způsobů jak soubor získat. Soubor se dá vytvořit programově a vložit mu obsah. Dá se získat požadavkem na server. Obrázek v elementu *canvas* se dá uložit do souboru. Soubor z lokálního počítače se dá dostat do webové aplikace pomocí formulářového prvku file nebo pomocí Drag & Drop API.

- Filesystem API
  API umožňuje vytvoření lokálního virtuálního filesystemu pro použití stránkou. Vhodné použití je cache souborů a ukládání dat, pro které není vhodný localstorage API.

- Fullscreen API
  API umožňuje programově přepnout do zobrazení na celou obrazovku a také aktuální způsob zobrazení detekovat pomocí definovaných událostí.

- Drag & Drop API
  API umožňuje virtuálně přesouvat prvky nejen přímo na stránce, ale i mezi stránkami a dokonce i z lokální adresářové struktury do stránky. Elementy mají nový atribut draggable a v Javascriptu je mnoho nových d&d událostí (dragStart, dragEnd, dragOver,...), které implementaci d&d velmi usnadňují oproti dřívějším implementacím pomocí jednodušších událostí myši. Při přetáhnutí souboru z lokální adresářové struktury do stránky má aplikace přístup k obsahu souboru.

- Webworksers - API pro běh na pozadí
  Javascript donedávna fungoval jen v jednom vlákně. Nyní existují takzvaní WebWorkeři, což jsou skripty, které mohou běžet na pozadí. WebWorkeři nemají přístup k samotné stránce a komunikují s hlavním vláknem pomocí zpráv.

- Websocket - API pro obousměrnou komunikaci klient-server
  Protokol HTTP je bezstavový, komunikace je vždy zahájena klientem a po odeslání požadavku se spojení ukončí. Pro určitý typ dat je tento způsob nepraktický, například kdyby klient potřeboval vědět o změně na serveru, musí se přes HTTP periodicky ptát, místo aby vyčkával na zprávu. Každý požadavek navíc vyžaduje režii (metadata požadavku i odpovědi). Jako řešení těchto problémů byl vytvořen WebSocket, což je obousměrná, spolehlivá, textová, asynchronní komunikace. Komunikace probíhá zasíláním zpráv, které spustí na straně příjemce callback funkci.

- API pro bitmapovou grafiku
  Toto api je součástí elementu *canvas* a umožňuje na něj kreslit bitmapovou grafiku, vkládat do něj obrázky, výsledný obrázek ukládat do souboru a mnoho dalšího.

- XMLHttpRequest2
  XMLHttpRequest2 je specifikace pro asynchronní požadavky na server. Tato vylepšená specifikace umožňuje mimo nových parametrů sledovat průběh uploadu v případě, že request posílá data.

CSS 3

CSS v HTML5 umožňuje další pokročilejší úpravy stránky jako:

- Animace
  Nyní je možné definovat animace jako změnu mezi 2 stavy (selektory). Stačí zadat jakým způsobem, jak dlouho a jaké CSS atributy se budou animovat. Animace může být například různou průhledností a velikostí elementu, pokud nad ním je myš (selektor :hover). Stačí nastavit délku animace, průběh animace (nejjednodušší průběh je lineární) a atributy opacity a height. Definují se jen okrajové stavy animace, prohlížeč má za úkol postarat se, aby přechod byl plynulý.

- Transformace
  V CSS je nyní možné provádět nad elementy mnoho transformací jako je otáčení, zmenšování, zploštění nebo i 3d otáčení.

- Sloupce
  CSS konečně umožňuje rozdělit obsah elementu do sloupců, to se doteď muselo dělat speciálními způsoby (tabulky, různé pozicované divy).

- Pokročilejší selektory
  Je možné identifikovat elementy dalšími specifičtějšími způsoby. (Každý n-tý element, pod element, element obsahující atribut, selektory specifické pro formulárové prvky.)

- Vestavěné fonty
  Nyní je možné mít na stránkách vlastní fonty a prohlížeč fonty použije, přestože samotný operační systém font nemá.

Cross-site HTTP requests

HTML5 zavádí do HTTP protokolu nové hlavičky jako součást CORS(cross-site origin requests), které umožňují cross-domain requesty - Javascriptové požadavky na jinou doménu, než na které Javascript běží. Tyto požadavky jsou standardně zakázané jako bezpečnostní riziko. Pokud ale server používá správné hlavičky, prohlížeč požadavek povolí.

Hlaviček je několik a každá povoluje část funkčnosti. Hlavní hlavička je hlavička *Access-Control-Allow-Origin*, která popisuje doménu s povolením posílat požadavky na server. Pokud se použije znak *, požadavek může zaslat kdokoliv.

Pokud dostane prohlížeč odpověď obsahující tuto hlavičku, může odpověď povolit. V některých případech (například odesílání velkého formuláře plného souborů) se neposílají všechna data a napřed se pošle HTTP požadavek s metodou OPTIONS, který má za úkol jen zjistit tyto hlavičky. Pokud přijdou povolující hlavičky, prohlížeč pošle původní request, jinak oznámí chybu. Jakmile prohlížeč získá informaci o tom, že může komunikovat, informaci si uloží a udržuje ji po dobu definovanou hlavičkou *Access-Control-Max-Age*, hodnota je čas v sekundách.

Povolený požadavek má stále svá omezení. Není možné používat některé HTTP hlavičky v požadavku, není možné používat Cookies a další.

Povolení dalších hlaviček požadavku je dané hlavičkou *Access-Control-Allow-Headers*. Hodnotou hlavičky jsou jednotlivé názvy hlaviček oddělené čárkou.

Povolení cookies je dané hlavičkou *Access-Control-Allow-Credentials*. Hodnota je *true* nebo *false*.

Povolení přístupu k hlavičkám odpovědi je dané hlavičkou *Access-Control-Expose-Headers*. Hodnotou jsou jednotlivé možné hlavičky oddělené čárkou.

Kromě zmíněných hlaviček ještě existují další, které popisují povolené metody a jiné.

2.4.2 Podpora v prohlížečích

Podpora zmíněných částí HTML5 je různorodá. Některé části jsou z většiny implementované, některé části jen některými prohlížeči, některé části nejsou implementovány vůbec a některé části pravděpodobně ze specifikace vypadnou. Části jsou často implementovány tak, že pokud se použijí ve stránce s prohlížečem, který je neumí, pořád fungují, i když ne tak komfortně. Formulářový prvek file s atributem multiple se chová jako by ho neměl, input formulářové prvky se chovají jako textové. Některé části samozřejmě fungovat nebudou vůbec, obzvlášť Javascriptové API jako je například Drag & Drop.

2.4.3 Podpora technologií vhodných pro KivFS

Ze zmíněného popisu HTML5 bych chtěl zdůraznit následující, protože se hodí pro implementaci klienta pro KivFS:

- Formulářový prvek file
  Formulářový prvek má momentálně možnost uploadovat více souborů najednou díky atributu multiple. To umožní uživateli komfortnější výběr souborů pro upload oproti starému způsobu, kdy musel uživatel každý soubor vybrat zvlášť.
  Attribut multiple je implementovaný ve většině moderních prohlížečů, v IE od verze 10.

- Drag & Drop API
  Drag & Drop API je nejen uživatelsky velmi přívětivé a přirozené, zároveň taky umožňuje přistupovat k souborům, které uživatel přetáhne z lokálního souborového systému do okna aplikace.
  Drag & Drop API je implementované ve většině moderních prohlížečů, v IE od verze 10.

- Filesystem API
  Filesystem API je skvělé pro implementaci klientské cache, momentálně je bohužel ve velmi experimentální fázi a z velkých prohlížečů má API implementovaný jen Chrome.

- Web storage
  Storage jsou úložiště, které umožňují ukládat data ve formě klíč-hodnota. Jsou vhodné pro ukládání informací u souborech v cache jako je verze souboru. Web storage je implementované ve většině moderních prohlížečů, v IE od verze 8.

- XMLHttpRequest2 XMLHttpRequest level 2 je nová verze objektu XMLHttpRequest, který vylepšuje možnosti volání serveru Javascriptem. Pomocí nové verze XMLHttpRequestu bude možné posílat binární data, posílat cross-origin request (tzn. request do jiné domény než kde běží Javascript, pokud to cílová doména povoluje) sledovat průběh uploadu a uživateli ho sdělovat.
  XMLHttpRequest je implementovaný ve většině moderních prohlížečů, v IE od verze 10.

2.5 Technologické požadavky na implementaci KivFS

Aby klient mohl komunikovat s KivFS, musí podporovat technologie, které používá KivFS ke komunikaci. Pokud některé technologie nebude možné v prostředí WWW použít, bude nutné použít náhradní řešení, aby se komunikace umožnila.

2.5.1 TCP

TCP je jeden ze základních spolehlivých protokolů, na kterém staví většina protokolů vyšších vrstev včetně KivFS.

V prostředí WWW není možné používat přímo, pro komunikaci se používá výhradně protokol HTTP nebo novější protokol WebSocket.

2.5.2 Kerberos

Kerberos je autentizační mechanismus používaný na celé Západočeské univerzitě pro autentizaci uživatelů orion konta. Mechanismus umožňuje autentizovat se bez nutnosti posílat po síti uživatelské heslo. Mechanismus sám o sobě řeší jen ověřování uživatele, neřeší ale autorizaci (práva).

V prostředí WWW není možné používat ani Kerberos přímo, musí být řešeno pomocí proxy na serveru. Existují řešení pro Kerberos do webových serverů, přihlašovací údaje napřed ale musí přijít na webový server a ten teprve zprostředkovává Kerberos autentizaci.

2.5.3 Šifrování

Se systémem KivFS se z důvodu bezpečnosti komunikuje KivFS protokolem zašifrovaným protokolem SSL.

V prostředí WWW se používá z důvodu bezpečnosti na šifrování protokol HTTPS, což je šifrovaná verze protokolu HTTP. Protokol WebSocket má také šifrovanou verzi. Pro komunikaci s webovým serverem bude jistě použita šifrovaná verze jednoho z protokolů.

2.5.4 Cache

Klient musí mít možnost dočasně ukládat soubory, aby mohl soubory cachovat a urychlovat tím práci se soubory, které se nezměnily.

Cache si musí kontrolovat konzistenci. Při každém požadavku na soubor se napřed klient podívá do cache, zda soubor je zde uložen. Pokud ano, zkontroluje, jestli je soubor aktuální dotazem na verzi souboru na serveru. Pokud verze souhlasí, soubor je aktuální.

V případě zaplněné cache se musí z cache některý ze souborů smazat, aby uvolnil místo pro nový soubor. Pro výběr tohoto souboru existuje velké množství strategií, které závisejí na faktorech jako je:

- Verze
  Číslo, které se inkrementuje při každé změně souboru na serveru.

- Četnost čtení
  Číslo, které se inkrementuje, kdykoliv bylo ze souboru na serveru čteno.
- Četnost zápisů
  Číslo, které se inkrementuje, kdykoliv bylo do souboru zapisováno.
- Datum poslední modifikace

Strategie zpravidla používají tyto hodnoty, aby využívaly cache maximálně efektivně - aby v cache zůstávaly soubory s největší šancí budoucí potřeby uživatele a přitom stále aktuální.

Základní používané strategie jsou tyto:

- Náhodný výběr
  Z cache se náhodně mažou soubory, dokud není dostatek místa pro nový soubor.
- FIFO
  First in, first out - z cache se mažou soubory, které jsou v cache nejdéle, dokud není dostatek místa pro nový soubor.
- FIFO second chance
  Stejně jako FIFO, ale soubory mají navíc příznak, který určuje, jestli byly od doby vložení do cache vyžadovány. Pokud soubor byl vyžadován, má přednost před souborem, který nebyl vyžadován, i když byl do cache vložen dříve.
- LIFO
  Last in, first out - z cache se mažou soubory, které jsou v cache nejkratší dobu, dokud není dostatek místa pro nový soubor.
- LRU
  Least recently used - v souborech se ukládá informace o času posledního použití. Z cache se odstraňují soubory, které mají nejstarší čas posledního použití, dokud není dostatek místa pro nový soubor.
- LFU
  Least frequently used - v souborech se ukládá informace o počtu použití. Z cache se odstraňují soubory, které mají nejmenší počet použití, dokud není dostatek místa pro nový soubor.

V implementaci jsem použil strategii LRU, protože metoda je jednoduchá na implementaci, a přesto efektivní.

2.6 Klient KivFS v prostředí WWW

K webovému klientu se dá přistupovat z několika pohledů v závislosti na způsobu použití a místě, kde bude hlavní část výkonného kódu. K tomuto se často vyskytují pojmy, jako je tlustý a tenký klient. Tenký klient je aplikace, která má serverovou část s logikou a klientskou, která jen posílá příkazy serveru, který kompletně vše zpracovává a vrací zpět na zobrazení. Typickým příkladem je klasická HTML stránka, kde veškeré soubory jsou na serveru, který je zasílá. Klient žádá o další soubory skrze odkazy. Opakem tohoto přístupu je případ Javascriptové aplikace, kde probíhá aplikační logika přímo v prohlížeči a server je volán jen v případě nutnosti získání dat nebo použití funkce. Server vrací vždy jen nutná data, celé soubory jen pokud je třeba velká změna nebo obnovení stránky.

2.6.1 Webové prohlížení adresáře namountovaného na serveru

V tomto případě by webový server měl distribuovaný systém připojený pomocí existujícího nativního klienta pro serverovou platformu a byl prohlížený webovým serverem - například by se na počítač s linuxem připojil (”namountoval”) KivFS do adresáře, který by byl prohlížitelný pomocí webového serveru Apache. Bylo by nutné jen zařídit přihlášení, provést autentizaci a připojení systému, který se bude následně zobrazovat a analogicky i odpojení systému při odhlášení. Toto řešení je nejjednodušší z hlediska implementace, postrádá však flexibilitu webové aplikace. Je závislé na platformě, kde běží webový server. S KivFS serverem bude komunikovat nativní klient serverové platformy, webová aplikace bude prohlížet adresář, který se bude pro aplikaci tvářit jako lokální. Toto řešení má charakter tenkého klienta.

2.6.2 Aplikace s logikou na webovém serveru

Aplikace bude mít hlavní část na serveru (J2EE, ASP.NET, atd.) a klient bude jen posílat příkazy pro server, webová aplikace bude komunikovat s KivFS serverem. Toto řešení má charakter tenkého klienta.

2.6.3 Aplikace s logikou u klienta

Hlavní logika by byla napsaná na klientské části, tzn. v Javascriptu, Java Appletu nebo jiné podobné technologii. Toto řešení má charakter tlustého klienta.

2.7 Shrnutí

Rozhodl jsem se umístit aplikační logiku do klienta v HTML5. Klient nemůže komunikovat přímo s KivFS, protože prostředí prohlížeče nepodporuje protokol TCP. Aby klient mohl komunikovat s KivFS a přitom měl plnil veškerou aplikační logiku, musí být vytvořena tenká proxy pro komunikaci s KivFS a protokol pro komunikaci mezi klientskou částí a proxy. Tímto způsobem bude vytvořena mezivrstva mezi klientem a serverem pro usnadnění práce a překlad HTTP požadavků na KivFS protokol a zpět.

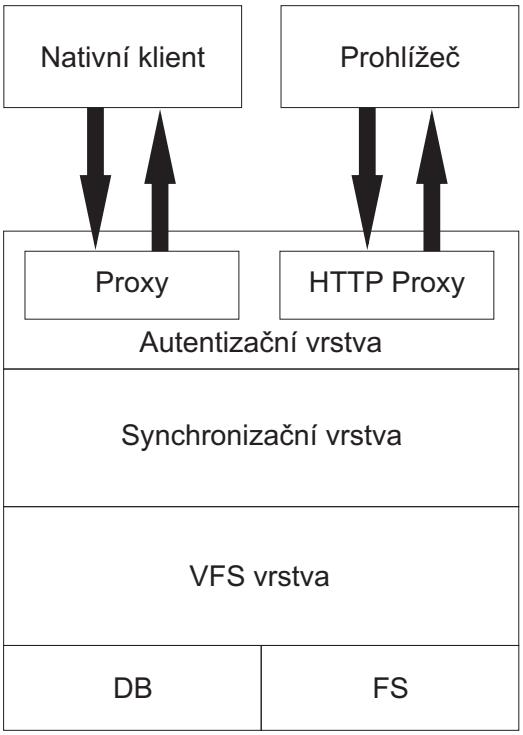
2.7.1 Proxy

Protože webové aplikace nemohou komunikovat s KivFS přímo (neumí TCP protokol), je nutné udělat proxy, které bude na jedné straně mít protokol, který bude umět webová aplikace. Na druhé straně bude propojení proxy-KivFS. Kromě TCP prohlížeč také neumí Kerberos, proto se přihlašovací údaje pošlou do proxy a proxy provede samotnou autentizaci proti Kerberu a KivFS. Z hlediska funkčnosti může takováto proxy splňovat funkčnost, kterou zastupuje KivFS proxy a proto není nutné KivFS proxy používat a vytvořit alternativní vstupní bod do KivFS s protokolem nad HTTP vedle binárního KivFS protokolu nad TCP.

Situaci popisuje obrázek 2.2.

Technologie

Proxy jsem se rozhodl implementovat v technologii Java EE, protože nabízí jednoduché API pro práci s HTTP a v Javě už je implementovaný protokol ve starší verzi z bakalářské práce.



Obrázek 2.2: Pozice HTTP Proxy z hlediska KivFS.

Výběr protokolu nižší vrstvy

Na straně prohlížeč-proxy jsou technologicky dostupné protokoly HTTP a WebSocket protokol. V tomto případě je vhodnější použít protokol HTTP, protože:

- Je podporovaný v každém prohlížeči.
- Je synchronní - ve formě požadavek-odpověď stejně jako protokol KivFS.
- Obsahuje hlavičky, které je možné převzít a použít.

Websocket je naopak méně vhodný, protože:

- Zatím neexistuje finální verze protokolu a specifikace se mění.
- Protokol není zdaleka podporovaný ve všech prohlížečích.

Autentizace

Protože prohlížeč neumí Kerberos, je nutné posílat jméno a heslo na server a ten teprve může udělat ověření KivFS autentizační proxy vrstvou.

Protokol klient - proxy

Nad protokolem HTTP se musí vytvořit protokol, který bude používat výsledná aplikace. Tento protokol bude nezávislý na webové aplikaci a bude možné udělat více aplikací využívajících tento protokol. Bude dokonce možné použít ho i v jiných prostředích, kde není možné použít TCP nebo Kerberos, nabízí se například klient pro Metro. Protokol bude nad protokolem HTTP a může proto používat některé jeho specifické hlavičky - například Accept pro rozlišení, v jakém formátu chce klient odpověď, hlavičku Range, pokud by klient chtěl jen část souboru, a další.

Cache

Protože bude použit tlustý klient, cache nemá smysl umist'ovat na proxy - cache musí být přímo u klienta. Takovou cache je teoreticky možné implementovat v HTML5, ale momentálně je podporována jen v prohlížeči Google Chrome, proto bude cache funkční jen v tomto prohlížeči.

3 Implementace

Implementace je rozdělená na 4 části. V první části je navržen a implementován protokol nad HTTP, který definuje operace s KivFS a jednoduchý HTML klient, který je schopný tento protokol využívat. V druhé je popsána implementace aplikace v HTML5, která plně využívá moderní technologie prohlížeče. Ve třetí části je popsána implementace aplikace v Javě EE, která využívá protokol a generuje jednoduchou HTML aplikaci, která je funkční i v nemoderních prohlížečích. V poslední části je popsána kombinace předchozích dvou aplikací a výhody této kombinace.

3.1 Proxy

Proxy je mezistupeň mezi KivFS webovým klientem a KivFS serverem, který zprostředkovává protokol nad HTTP pro webovou aplikaci. Aby proxy mohla převádět příkazy z jednoho protokolu do druhého, musí oběma protokolům rozumět.

Na straně proxy-KivFS jsem vyšel ze své bakalářské práce, kterou jsem musel upravit, aby implementovala novou verzi KivFS protokolu a nové příkazy.

3.1.1 Protokol prohlížeč - proxy

V následujících sekcích jsou popsány jednotlivé příkazy, formáty zpráv a odpovědí. Protokol byl navržen tak, aby používal co nejvíce částí přímo z HTTP protokolu, některé hodící části se přesto nepoužily. Například HTTP metody PUT nebo DELETE nebyly použity pro jejich logické využití, protože HTML formuláře neumí tyto metody používat a to by snižovalo rozsah použití protokolu. Místo toho je všude použita metoda GET, jen v případě uploadu souborů je použita metoda POST. Byly ale použity některé hodící se hlavičky HTTP protokolu pro požadavek:

- Accept
  Hlavička pro rozlišení požadovaného typu návratové hodnoty - json

pro Javascriptovou aplikaci (hodnota application/json) nebo HTML pro jednoduchou HTML odpověď (hodnota text/html).

- Authorization
  Hlavička pro poslání dat pro autorizaci, posílají se v ní login údaje ve formátu jméno:heslo zakódované pomocí Base64.

Pro odpověď bylo použito několik hlaviček:

- WWW-Authenticate
  Oznámení klientovi, že se musí nejprve přihlásit.
- Access-Control-Allow-Headers, Access-Control-Allow-Origin, Access-Control-Max-Age
  Tyto hlavičky umožňují Javascriptu komunikovat i s jinou doménou, než na které běží (cross-domain requests).

Společné pro všechny

Parametry v POST/GET: *sessionId*: Hodnota aktuálního připojení do KivFS, bez této hodnoty nelze používat žádné příkazy, kromě příkazu connect, který *sessionId* získává.

SessionId není součástí cookies, protože umožňuje útoky typu CSRF (cross-site request forgery) a obecně jsou cookies často terčem útoků.

Parametry v hlavičce:
Accept: application/json

Popis: Pokud se vyskytuje tento parametr, odpověď bude ve formátu json místo HTML - vhodné pro aplikace, které chtějí data ve formátu, který je lepší ke zpracování.

Authorization: AUTH

Popis: Tato hlavička posílá přihlašovací údaje ve formě *LOGIN:PASSWORD*, zakódované pomocí Base64. Kódování je z důvodu jednoduššího přenosu, protože výsledný Base64 řetězec obsahuje jen ascii znaky. Z bezpečnostního hlediska je potřeba heslo zabezpečit jiným způsobem - na to se používá šifrovaná verze protokolu HTTP - HTTPS.

V případě jakékoliv chyby příkazu:
Chybová stránka s popisem chyby nebo json data obsahující popis chyby, pokud požadavek obsahoval "Accept: application/json".

V případě nepřihlášení uživatele se vrátí uživateli s odpovědí hlavička WWW-Authenticate, která klientovi oznamuje, že se musí napřed přihlásit.

Připojení

`URL: /connect/?kivfsHostname=HOSTNAME&kivfsPort=PORT`

Návratová hodnota:
Hodnotu sessionId používané dále při každém requestu, které identifikuje otevřené připojení.

Popis funkce:
Připojí se k KivFS na hostname HOSTNAME a port PORT s přihlašovacími údaji LOGIN a PASSWORD.

Odpojení

`URL: /disconnect/`

Návratová hodnota:
Stránka informující o úspěchu nebo jednoduchá json struktura oznamující úspěch.

Popis funkce:
Provede odhlášení z KivFS.

Výpis adresáře

`URL: /ls/CESTA`

Návratová hodnota:

1. HTML
   Stránka s výpisem adresáře v případě HTTP odpovědi. Jednotlivé soubory jsou popsány v tabulce, mají nastavené třídy pro rozlišení a každý soubor odkazuje odkaz, který v případě adresáře vede na vylistování daného adresáře a v případě souboru podnikne jeho stáhnutí.
2. JSON
   Přehlednou strukturu obsahující informace o souborech v adresáři.

Popis funkce:
Vrátí seznam souborů adresáře v parametru CESTA.

Vytvoření adresáře

`URL: /mkdir/CESTA`

Návratová hodnota:
Stránka informující o úspěchu nebo jednoduchá json struktura oznamující úspěch.

Popis funkce:
Vytvoří adresář CESTA.

Smazání adresáře

`URL: /rmdir/CESTA`

Návratová hodnota:
Stránka informující o úspěchu nebo jednoduchá json struktura oznamující úspěch.

Popis funkce:
Smaže adresář CESTA.

Smazání souboru

`URL: /rm/CESTA`

Návratová hodnota:
Stránka informující o úspěchu nebo jednoduchá json struktura oznamující úspěch.

Popis funkce:
Smaže soubor CESTA.

Dávkové mazání

`URL: /rmmix/CESTA?path=CESTA&path=CESTA&path=...`

Návratová hodnota:
Stránka informující o úspěchu nebo jednoduchá json struktura oznamující úspěch.

Popis funkce:
Smaže všechny adresáře v jednom požadavku. Pokud v průběhu mazání dojde k chybě, zpracování se ukončí a vrátí se chybová hláška.

Přesunutí souboru

`URL: /mv/?from=FROM&to=TO`

Návratová hodnota:
Stránka informující o úspěchu nebo jednoduchá json struktura oznamující úspěch.

Popis funkce:
Přesune soubor/adresář z cesty FROM do cesty TO.

Upload souboru

`URL: /put/CESTA`

Návratová hodnota:
Stránka informující o úspěchu nebo jednoduchá json struktura oznamující úspěch.

Popis funkce:
Uploaduje do adresáře CESTA všechny soubory, které byly poslány v HTTP POST požadavku, se stejnými názvy jako byly v požadavku.

Download souboru

`URL: /get/CESTA?path=CESTA&path=CESTA&path=...`

Návratová hodnota:
Požadovaný soubor nebo zip archiv obsahující všechny požadované soubory.

Popis funkce:
Pokud je jen jeden parametr CESTA, stáhne přímo soubor. Pokud je parametrů více, pošle všechny soubory v jednom zip archivu (protokol HTTP standardně neumí posílat více souborů v 1 odpovědi).

Shrnutí

Příkaz	URL	Metoda
Připojení	/connect/?kivfsHostname=H&kivfsPort=P	GET
Výpis adresáře	/ls/CESTA	GET
Vytvoření adresáře	/mkdir/CESTA	GET
Smazání adresáře	/rmdir/CESTA	GET
Smazání souboru	/rm/CESTA	GET
Dávkové mazání	/rmmix/CESTA?path=P1&path=2&...	GET
Přesunutí	/mv/?from=FROM&to=TO	GET
Upload	/put/CESTA	POST
Download	/get/CESTA&path=CESTA	GET
Odpojení	/disconnect/	GET

Tabulka 3.1: Seznam příkazů protokolu KivFS HTTP.

3.1.2 Autentizace s KivFS

Při poslání údajů na server (příkaz connect) proxy napřed jméno a heslo ověří protokolem Kerberos. Pokud údaje souhlasí, proxy se připojí na KivFS

server a pokusí se nastavit v KivFS uživatele (příkaz KIVFS_ID). V případě, že uživatel v KivFS existuje, autentizace je dokončena a klient může začít používat KivFS.

3.1.3 Session a bezpečnost

V případě úspěšné autentizace si vytvoří proxy náhodný řetězec *sessionId* pro komunikaci mezi proxy a www klientem a spojí si k *sessionId* připojení na KivFS. K danému připojení se poté může připojit jen klient se stejným sessionId. Komunikace mezi proxy a klientem bude probíhat šifrovaně (SSL) a komunikace mezi prohlížečem a proxy bude také probíhat šifrovaně (HTTPS), což zaručí bezpečnost *sessionId* a zabrání jejímu ukradení. V případě delší nečinnosti se spojení daného *sessionId* ukončí.

3.1.4 Realizace protokolu

Úprava KivFS knihovny

Aby bylo možné komunikovat s KivFS, bylo napřed třeba změnit knihovnu z mé bakalářské práce a zapracovat všechny změny v protokolu. V bakalářské práci jsem použil test driven development, proto jsem musel napřed vytvořit testy odpovídající novému protokolu, a pak jednotlivé části protokolu implementovat.

Implementace

K protokolu jsem použil technologii servlet api z J2EE na parsování HTTP protokolu a rozhraní pro HTTP server.

Aplikace je rozdělena na 2 hlavní obslužné servlety. První servlet - *KivfsCommandsServlet*, je pro obsluhování jednotlivých příkazů - zpracovává protokol, příkazy a odesílá odpovědi. Druhý servlet - *Html5PathServlet*, je servlet, který generuje verzi aplikace, která potřebuje minimum Javascriptu. Tento servlet generuje dokument výpisu adresářů, u každého adresáře jsou odkazy na možné akce. Odkazy na možné akce přesměrovávají na první servlet, který zobrazí výsledek akce a odkaz na návrat zpět na výpis souborů.

Takto vytvořený dokument je možné ovládat a používat i v prohlížečích jako je *links* nebo i ve starších mobilních zařízeních. HTML5 aplikace druhý zmíněný servlet nepotřebuje a je vytvořen jen jako fallback řešení pro prohlížeče, které HTML5 nepodporují. Pro oba servlety existuje společný předek, který sdružuje společné funkce, jako je získávání KivFS spojení, Kerberos autentizace a další.

Při prvním požadavku na připojení a přihlášení do KivFS si server vytvoří KivFS spojení, a to vloží do struktury napojené na *sessionId*. Při dalších požadavcích se už bere aktuální KivFS spojení a na to se přeposílají příkazy.

Server buď odpovídá v jednoduchých json strukturách nebo jednoduchých HTML stránkách (v závislosti na parametru Accept v hlavičce požadavku). HTML odpovědi jsou zobrazeny technologií JSP.

3.1.5 Jednoduchý klient v HTML a Javascriptu

Výše popsaný protokol se dá pak jednoduše naimplementovat v HTML s minimem Javascriptu. Vystačíme si s HTML formuláři a občasným Javascriptem. Pokud chceme, aby cesta nebyla jako GET parametr, ale přímo v url, jako u příkazů mkdir/rmdir/rm, bez Javascriptu se neobejdeme. Například jednoduchý formulář pro vytvoření adresáře můžeme implementovat takto:

```
<form id="form1" method="get">
<input id="mkdirPath" />
<input type="submit" value="Vytvor adresar"
onclick="document.getElementById('form1')
.action='http://147.228.63.63:8080/httpserver/mkdir'
+ document.getElementById('mkdirPath').value;" />
</form>
```

Pro každý typ požadavku stačí vytvořit takovýto formulář a nejjednodušší klient je hotový.

Výhoda takto vytvořeného klienta je, že prakticky jakýkoliv prohlížeč s minimální podporou Javascriptu bude schopný klienta používat včetně textových prohlížečů jako je links. Stejně tak s takovýmto protokolem je možné implementovat klienta pro další platformy, které nepodporují TCP protokol

nebo Kerberos a přitom podporují HTTPS protokol, jako například platforma Metro.

Takovýto klient je sice jednoduchý a plně funkční, není ale příliš uživatelsky přívětivý (blíží se používání příkazové řádky).

3.2 Webová aplikace v HTML5

Webová aplikace je stěžejní část celé práce. HTML5 nabízí pro uživatele více možností jak pracovat se soubory a prohlížečem, aby se aplikace ovládala co nejsnadněji a zároveň fungovala na co nejvíce platformách.

3.2.1 Použité Javascriptové knihovny

Samotné HTML5 ale neřeší některé další věci, pro které jsem použil hotové knihovny. Knihovny jsou stažené v projektu httpserver v adresáři *src/main/webapp/js*.

- Jednotné chování prohlížečů v určitých případech
  Pro jistější fungování některých funkčností a zároveň zjednodušení práce s Javascriptem a DOM jsem použil hodně používanou knihovnu jQuery.
- Vytváření částí HTML stránek pomocí Javascriptu
  Javascript sice umí vytvářet části HTML, ale ne velmi úhledným způsobem, proto jsem použil templatovací nástroj dust.js, který umožňuje vytvářet části HTML jako sloučení dat a šablon.
- Řazení v tabulkách
  Na řazení v tabulkách jsem použil vhodnou knihovnu tablesorter, což je modul do už používané knihovny jQuery. Knihovna umožňuje definovat, které sloupce tabulek je možné řadit a jakým způsobem.
- Parsování a normalizace URI
  Na parsování a normalizaci URI jsem použil knihovnu URI.js, která má vhodné metody pro úpravu URI.
- Unit testování
  Pro unit testování jsem použil knihovnu gunit.js, která má vhodné api pro unit testování a obsahuje i podporu pro asynchronní testování.

3.2.2 Vzhled

Základem vzhledu je sémantický obsah, který tvoří z větší části tabulka obsahující seznam souborů. Tabulka je vytvářena templatovacím frameworkem dust.js. Dynamické části stránky jsou logicky rozdělené na menší komponenty - šablony, které jsou generované nástrojem dust. Příklad šablony, která generuje 1 řádku v tabulce souborů, vypadá následovně:

```
<tr class="type{type}" id="{path}{name}/"
data-fullpath="{path}{name}/"
data-fullname="{path}{name}" data-path="{path}"
data-type="{type}" data-name="{name}"
title="Název: {name} &#10;Velikost: {size_display.size}
{size_display.units} &#10;
Vlastník: {owner} Skupina: {group} &#10">
<td class="selection"><input id="option_{path}{name}"
{attributes} data-path="{path}"
data-type="{type}" type="checkbox" name="path"
value="{path}{name}" /></td>
<td class="name"><a href="get{path}{name}">{name}</a></td>
<td class="size" data-sort-val="{size}"><span class="value">
{size_display.size}</span>
<span class="units">{size_display.units}</span></td>
...
```

Spojením šablony a konkrétních dat ve formátu json vznikne HTML kód, který tvoří výsledný vzhled. Příklad části dat, který koresponduje s předchozí šablonou, může vypadat takto:

```
{
  name:"soubor.txt",
  path:"/",
  type:"1",
  size_display: {
    size: 1,
    units: "KB"
  },
  size: 1024,
  owner: "root",
```

```
  group: "root",
  ...
}
```

Výsledné spojení může vypadat takto:

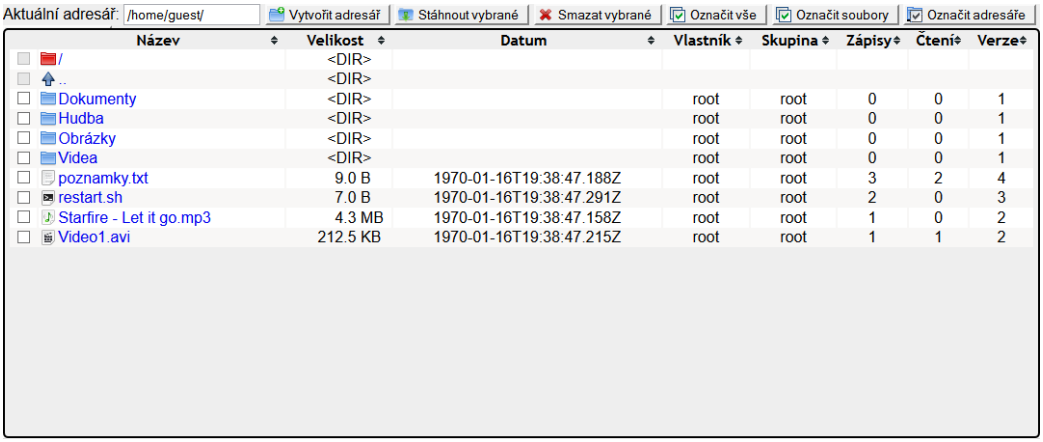
```
<tr class="type1" id="/soubor.txt"
data-fullpath="/soubor.txt/"
data-fullname="/soubor.txt" data-path="/"
data-type="1" data-name="soubor.txt"
title="Název: soubor.txt &#10;Velikost: 1
KB &#10;
Vlastník: root Skupina: root &#10">
<td class="selection"><input id="option_/soubor.txt"
data-path="/"
data-type="1" type="checkbox" name="path"
value="/soubor.txt" /></td>
<td class="name"><a href="get/soubor.txt">soubor.txt</a></td>
<td class="size" data-sort-val="1024"><span class="value">
1</span>
<span class="units">KB</span></td>
...
```

Kromě tabulky je na obrazovce ještě horní ovládací panel s ovládacími tlačítky.

Celkový vzhled je přizpůsobený pomocí kaskádových stylů CSS. Tlačítka mají každé reprezentující ikonu pro lepší přehlednost. Každý soubor má také ikonu reprezentující jeho obsah podle typu přípony, která také zlepšuje celkovou přehlednost stránky.

3.2.3 Cache

Cache je implementována pomocí Filesystem API a Localstorage a momentálně funkční jen v prohlížeči Google Chrome, protože ten jediný implementuje Filesystem API. V lokálním filesystému jsou uložené soubory a v localstorage jsou uložené informace o souboru jako je verze, readhits a writehits. Klíčem je absolutní cesta souboru a hodnotou je jeho verze. V případě



Název	Velikost	Datum	Vlastník	Skupina	Zápisy	Čtení	Verze
/	<DIR>						
..	<DIR>						
Dokumenty	<DIR>		root	root	0	0	1
Hudba	<DIR>		root	root	0	0	1
Obrázky	<DIR>		root	root	0	0	1
Videa	<DIR>		root	root	0	0	1
poznamky.txt	9.0 B	1970-01-16T19:38:47.188Z	root	root	3	2	4
restart.sh	7.0 B	1970-01-16T19:38:47.291Z	root	root	2	0	3
Starfire - Let it go.mp3	4.3 MB	1970-01-16T19:38:47.158Z	root	root	1	0	2
Video1.avi	212.5 KB	1970-01-16T19:38:47.215Z	root	root	1	1	2

Obrázek 3.1: Prohlížení adresáře v prohlížeči Firefox 20.0.

downloadu jednoho souboru se napřed klient podívá, jestli neexistuje soubor v cache. Pokud existuje soubor před downloadem, dotáže se klient serveru na poslední verzi. Pokud verze souhlasí, klient poskytne verzi z cache. Pokud verze nesouhlasí, klient soubor stáhne, přepíše, aktualizuje cache a nabídne soubor uživateli. V případě stahování adresáře nebo více souborů najednou se cache nepoužívá, protože v těchto případech se posílají data jako 1 zazipovaný soubor. Zazipovaný soubor se nikdy necachuje, protože se nepředpokládá, že by uživatel stahoval stejnou kombinaci souborů se stejnou verzí.

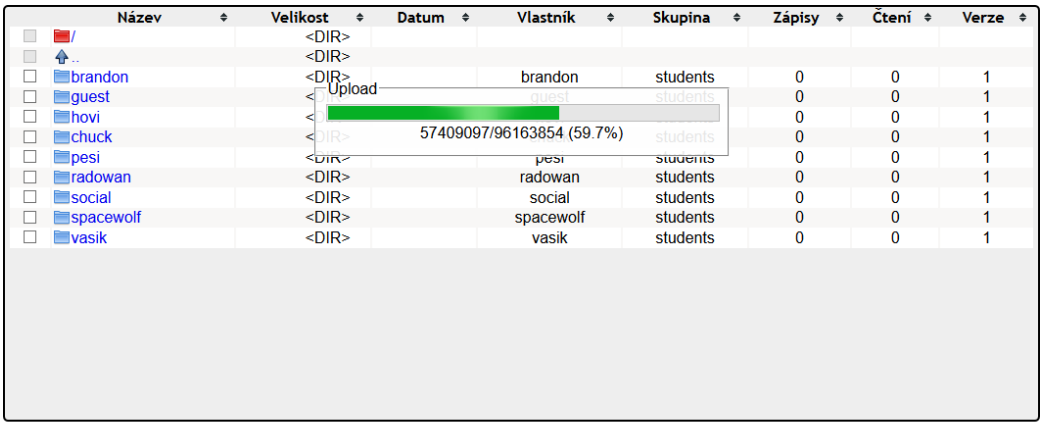
Cache funguje transparentně nezávisle na uživateli - pokud je dostupná, tak se zapne. Uživatel může cache zaznamenat jedině zrychlením opakovaného stahování souborů.

Aktuální obsah cache se dá zobrazit na adrese:
filesystem:http://www.example.com/temporary/

Adresa musí být zobrazena v prohlížeči Google Chrome a www.example.com musí být nahrazeno doménou kde běží KivFS klient.

3.3 Serverová HTML aplikace

V případě nedostatečné podpory klientského prohlížeče by uživatel nebyl schopný používat KivFS www klienta. Pro tento případ byla vytvořena ser-



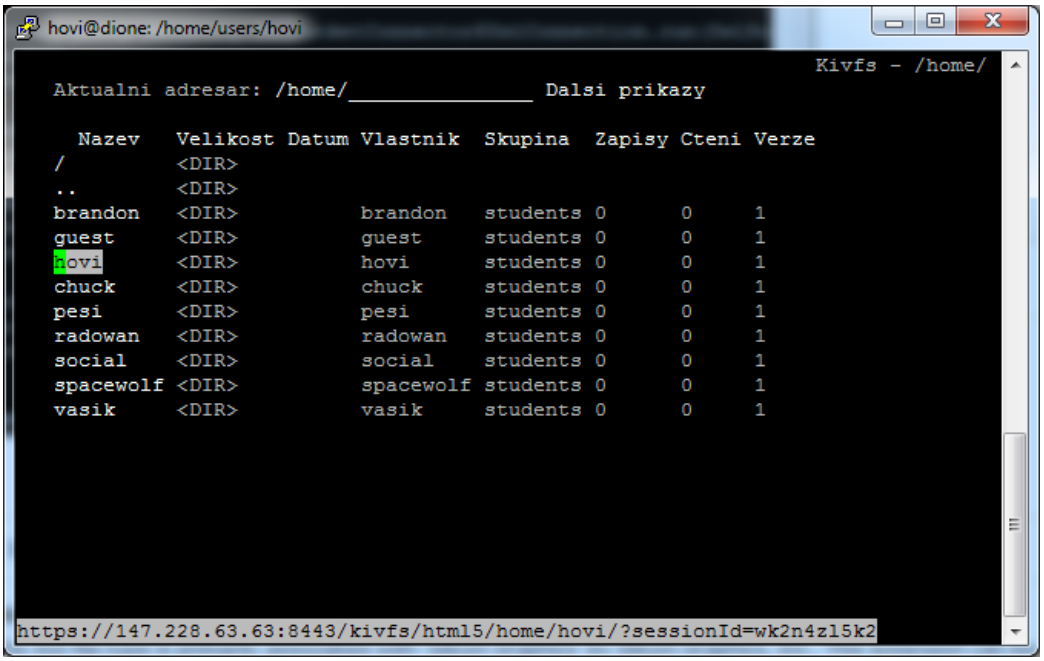
Obrázek 3.2: Prohlížení upload souboru v prohlížeči Firefox 20.0.

verová aplikace jako ”fallback”řešení, které bude fungovat i v nejjednodušších prohlížečích. Většina příkazů protokolu vyžaduje HTTP GET metody - pro provedení příkazu stačí navštívit danou adresu. Výjimkou je příkaz pro upload, kde je použita metoda POST. Posledním nestandardním příkazem je výpis adresáře a to je stěžejní část této aplikace. Aplikace při puštění příkazu výpisu s adresáře a požadavku na odpověď v HTML formátu generuje HTML tabulku (vzhledově a strukturou velmi podobnou tabulce, kterou používá HTML5 aplikace), která obsahuje soubory adresáře a odkazy pro operace s nimi, které vedou na jednotlivé url adresy příkazů. Takovou tabulku s odkazy je schopný zobrazit a interpretovat jakýkoliv prohlížeč.

Výhodou takového řešení je vysoká kompatibilita, nevýhodou je nižší úroveň uživatelského rozhraní.

3.4 Kombinace serverové a HTML5 aplikace

Výsledný kompromis obou aplikací je serverová aplikace, která zároveň obsahuje HTML5 aplikace. Pokud prohlížeč podporuje HTML5, spustí se HTML5 aplikace, která ovládá webovou aplikaci a překryje HTML vygenerované serverovou aplikací. Pokud prohlížeč nepodporuje HTML5, uživatel uvidí webovou aplikaci vygenerovanou serverovou aplikací. Upload a přesouvání jsou realizované samostatnou obrazovkou s formulářem.



Obrázek 3.3: Výpis adresáře v textovém prohlížeči links.

Tímto způsobem je výsledná aplikace kombinace obou a uživateli se vždy pustí ta aplikace, která bude schopná běhu.

3.4.1 Struktura projektů a spuštění

Projekt je rozdělen na 2 podprojekty, každý spravovaný nástrojem *maven*. Maven je nástroj, který definuje strukturu projektů v závislosti na jejich povaze. Díky jednotné struktuře se nemusí složitěji konfigurovat typické akce (goaly) pro projekt jako je třeba kompilace, deploy a podobně.

První projekt je Java knihovna pro komunikaci s KivFS. Obsahuje veškerý kód pro komunikaci s KivFS, Kerberos autentizaci a testy.

Java knihovna

Tento projekt je v adresáři *core* a jeho struktura má standardní formu *maven* Java projektů. Zdrojové kódy jsou v adresáři *src*. V adresáři *src* jsou podad-

resáře *main* pro výkonný kód a *test* pro testy. V každém z podadresářů jsou adresáře *java* pro zdrojové kódy v jazyce Java a *resources* pro všechny ostatní soubory (hlavně konfigurační). Při prvním spuštění se v kořenovém adresáři vytvoří adresář *target* se zkompilovanými soubory, případně s výsledky testů. Pokud se spustí část kódu, která generuje výstupy, veškeré logy se nacházejí ve vytvořeném adresáři *log*.

Knihovna se sama nepouští. Slouží ke spouštění testů a také je využívána druhým projektem.

Testy se spouští příkazem:

```
mvn test
```

Konfigurace testů je v souboru *src/test/resource/test.properties*.

Pro instalaci knihovny do maven repozitáře, aby ji mohl používat druhý projekt, slouží příkaz (před instalací se také spustí testy):

```
mvn install
```

Překlad bez spuštění testů se spustí příkazem:

```
mvn -DskipTests install
```

Webová aplikace

Tento projekt je v adresáři httpserver a jeho struktura má standardní formu *maven j2ee-simple* Java projektů. Struktura je stejná jako u Java projektů s přidaným adresářem *src/main/webapp*, který obsahuje část webové aplikace pro klientskou část (JSP, HTML, Javascript, CSS, obrázky,...) a je to kořenový adresář webové aplikace. V tomto adresáři je adresář *js*, který obsahuje kompletní aplikační logiku HTML5 aplikace, adresář *css*, který obsahuje kaskádové styly serverové i HTML5 aplikace, adresář *img*, který obsahuje obrázky, a adresář *WEB-INF*, který obsahuje soubor *web.xml*, který používá servletový kontejner pro mapování adres na servlety. V kořenovém adresáři webové aplikace jsou dále JSP soubory, které používá serverová aplikace pro

generování odpovědí klientovi a HTML soubory, které používá HTML5 aplikace.

Spuštění proxy probíhá příkazem:

```
mvn package jetty:run
```

Příkaz vytvoří *war* archiv, který následně spustí v servletovém kontejneru jetty. Toto spuštění je jen pro testovací účely a spouští server s vygenerovaným SSL certifikátem pro HTTPS spojení na portu 8443. Pro použití jiného certifikátu v jetty je třeba změna konfigurace *maven* v souboru *pom.xml*, který se nachází v kořenovém adresáři celého projektu. Soubor *pom.xml* slouží i pro další konfiguraci proxy. Samozřejmě je možné ale použít jakýkoliv jiný kontejner jako je Tomcat[Jason Brittain(2003)], Winstone, JBoss, Glassfish a jiné. Jejich konfigurace a spouštění se liší. Serverová i HTML5 aplikace jsou po spuštění dostupné na adrese:
https://localhost:8443/kivfs/

Pro spuštění samotné HTML5 aplikace není nutné pouštět celý projekt. Stačí otevřít soubor *app.html* v kořenovém adresáři webové aplikace a zadat potřebné údaje pro připojení ke KivFS. Pro běh HTML5 aplikace je ale nutné HTTP proxy, které se při zadávání zadává.

4 Testy a měření

V této kapitole se zabývám testováním funkčnosti a měřením výkonu. Kapitola je rozdělená na část Java knihovny, která komunikuje přímo s KivFS serverem a na část Webová aplikace, která komunikuje s KivFS serverem skrz proxy. V obou případech se testuje funkčnost a měří výkon.

4.1 Java knihovna

Java knihovna je implementace KivFS protokolu, která je použitá na proxy pro komunikaci s KivFS serverem.

4.1.1 Testy funkčnosti

Pro testování funkčnosti jsou použity jUnit[Vincent Massol(2003)] testy. Knihovna nejde standardně ani zkompilovat bez toho, aby testy prošly. Tyto testy obsahují testy základních operací s KivFS, jako je vytvoření adresáře, smazání adresáře, vytvoření souboru, smazání souboru, upload souboru a download souboru. Tyto testy nejen že kontrolují funkčnost knihovny, ale zároveň ověřují i funkčnost serveru.

4.1.2 Zátěžové testy

Zátěžové testy jsou pro ověření správné funkčnosti serveru v náročnějších podmínkách. Testuje se zde mnohonásobný přístup, rekurzivní vytváření velké zanořené adresářové struktury a další testy jako kontrolu správného kódování. Díky některým z těchto testů se podařilo na serveru odstranit několik chyb.

4.1.3 Měření

Poslední část testů Java knihovny se soustřeďuje na měření výkonu v několika scénářích. Všechny testy proběhly s klientem běžícím na stroji ares.fav.zcu.cz a serverem na 147.228.67.124. Mezi stroji je 100Mbitové spojení.

Klientský počítač (ares.fav.zcu.cz) měl tyto parametry:

```
Procesory: 4x Intel(R) Xeon(TM) CPU 3.40GHz
Operační paměť: 32GB
Operační systém: Debian linux, jádro 3.2.0-2-amd64
Souborový systém s testovacími daty: ext4
Síťové připojení: 1Gbit
```

Upload a download 1GB souboru

Cílem tohoto testu je uploadovat soubor na server, následně ho stáhnout zpět, ověřit správnost souboru a změřit rychlost přenosu. Tabulka 4.1 obsahuje naměřené hodnoty přenosů.

měření	1	2	3	4	průměr
rychlost uploadu (MB/s)	42,1	41,5	43,2	40,1	41,7
rychlost downloadu (MB/s)	20,4	21,0	20,8	21,0	20,8
doba uploadu (s)	25,5	25,8	24,8	26,7	25,7
doba downloadu (s)	52,4	50,9	51,6	50,9	51,4
doba režie (s)	3,0	3,2	3,5	2,3	3,0
celková doba testu (s)	81	80	80	80	80,25
podíl režie na době (%)	3,75	4,06	4,43	2,92	3,79

Tabulka 4.1: Měření uploadu a downloadu 1GB souboru.

V mé bakalářské práci byla rychlost downloadu a uploadu na hranici propustnosti 100Mbit sítě, protože byla měřená na 100Mbit síti a proto není přesné srovnání samotných měření. Bakalářskou práci jsem ale také krátce testoval na 1Gbit síti a maximální rychlost uploadu byla 26MB/s, a proto je i bez srovnání samotných měření znatelný nárůst výkonu. Díky větší propustnosti sítě a zrychlení serveru z hlediska posílání dat je nyní režie v průměru 3,79%, což je mnohem méně než v mé bakalářské práci, kde se pohybovala

režie kolem 22%. Rychlost uploadu je cca 4x větší než v bakalářské práci, rychlost downloadu cca 2x větší.

Upload a download adresáře malých souborů

Zadáním testu byl upload adresáři s 512 soubory, každý o velikosti 128KB.

měření	1	2	3	4	průměr
rychlost uploadu (MB/s)	40,5	42,5	40,5	41,7	41,3
rychlost downloadu (MB/s)	14,0	13,2	13,8	14,0	13,8
doba režie (s)	596,7	603,5	612,7	603,8	604,2
celková doba testu (s)	603,1	610,3	619,1	610,7	610,8
podíl režie na době (%)	98,96	98,94	98,98	98,98	98,96

Tabulka 4.2: Měření uploadu a downloadu 512 souborů o velikosti 128KB.

Oproti bakalářské práci se sice zrychlil samotný přenos, přenos pro každý soubor ale trvá řádově v milisekundách. Příprava a režie pro přenos souboru zabírá skoro 99%, což je ještě více než v bakalářské práci. Důvodem zvýšení režie je zesložitění protokolu. Ve staré verzi protokolu se musí poslat 1 požadavek na přenos, jako odpověď přišla adresa na socket, na který se data odeslala a socket se zavřel. V nové verzi protokolu poslat požadavek na otevření souboru, poté požadavek na přenos, který jako odpověď pošle adresu na socket, na který se data odešlou a socket se zavře. Poté se ještě musí poslat příkaz flush a příkaz close, což je dohromady o 3 KivFS zprávy s odpověďmi více. Tento nárůst příkazů protokolu má za následek dokumentované zpomalení.

Upload a download adresářové struktury s malými soubory

Zadáním testu byl upload 5 adresářů s 512 soubory, každý soubor o velikosti 128KB.

Naměřené hodnoty jsou srovnatelné s předchozím testem.

měření	1	2	3	4	průměr
rychlost uploadu (MB/s)	40,4	40,3	40,7	43,4	41,2
rychlost downloadu (MB/s)	13,4	14,9	14,9	12,5	13,9
doba režie (s)	3138,9	3057,1	3034,5	3061,6	3073,0
celková doba testu (s)	3171,3	3087,2	3064,4	3095,2	3104,5
podíl režie na době (%)	98,98	99,02	99,02	98,91	98,98

Tabulka 4.3: Měření uploadu a downloadu adresářové struktury.

4.2 Webová aplikace

4.2.1 Základní funkčnost v prohlížeči

Základní funkčnost webové aplikace je testována pomocí unit testů podobně jako KivFS Java klient. Testy slouží k testování knihovny a tříd vytvořených v Javascriptu, které implementují KivFS HTTP protokol. Velký rozdíl oproti testům v Javě je jejich konstrukce, protože Javascript pracuje při HTTP komunikaci asynchronně. Tyto testy slouží k testování funkčnosti proxy a knihoven, nekontrolují ale samotné grafické rozhraní. Grafické rozhraní proběhlo pouze ručně. Pro testování jsem použil framework qunit.js.

Test je možné spouštět jen v prohlížečích, které umí základní File API, test byl spuštěn v následujících prohlížečích a platformách:

Prohlížeč	Verze	Výsledek
Firefox	20.0.1	100% funkčnost
Google Chrome	26.0.1410.64	100% funkčnost
Opera	12.12	100% funkčnost
Safari	5.1.7	100% funkčnost

Tabulka 4.4: Výsledky testování funkčnosti prohlížečů v OS Windows 7 64bit.

Prohlížeč	Verze	Výsledek
Firefox	20.0.1	100% funkčnost
Google Chrome	26.0.1410.64	100% funkčnost
Opera	12.12	100% funkčnost
Internet Explorer	10.0.9200.16540	100% funkčnost
Internet Explorer, mode IE9	10.0.9200.16540	módu IE9 nefunkční

Tabulka 4.5: Výsledky testování funkčnosti prohlížečů v OS Windows 8 32bit.

Prohlížeč	Verze	Výsledek
Firefox	20.0	100% funkčnost
Google Chrome	26.0.1410.63	100% funkčnost
Chromium	25.0.1364.160	100% funkčnost

Tabulka 4.6: Výsledky testování funkčnosti prohlížečů v OS Ubuntu linux 12.04.

Prohlížeč	Verze	Výsledek
Firefox	18.0.2	100% funkčnost
Google Chrome	26.0.1410.64	100% funkčnost
Opera	12.15	100% funkčnost
Internet Explorer	7.0.6001.18000	nefunkční

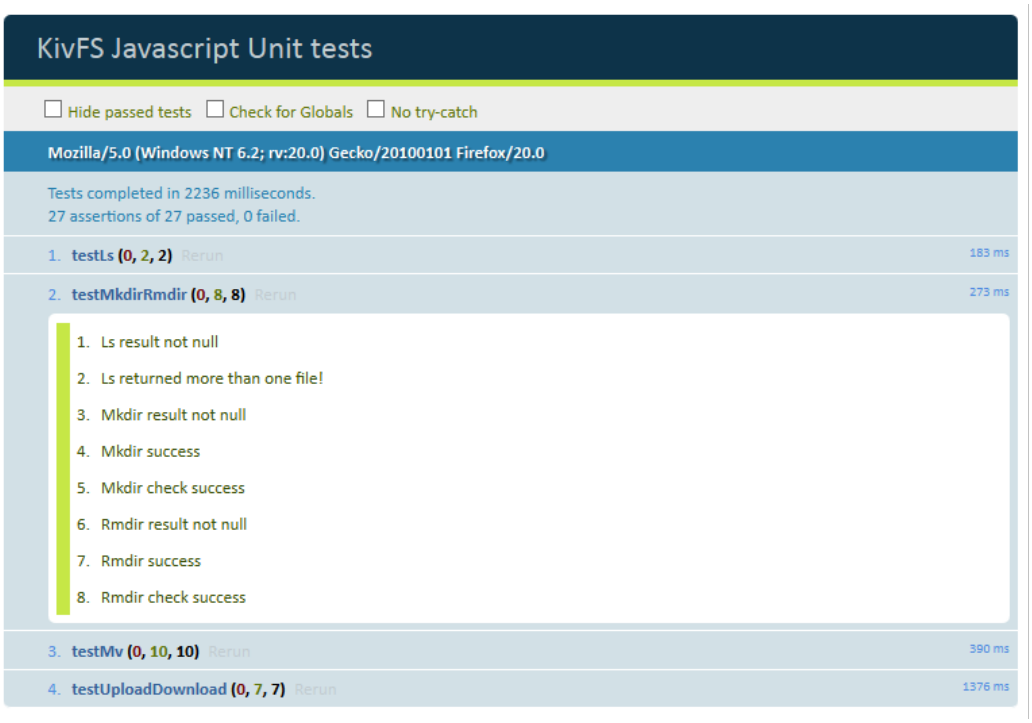
Tabulka 4.7: Výsledky testování funkčnosti prohlížečů v OS Windows Vista 32bit.

4.2.2 Podpora prohlížečů

Testování prohlížečů proběhlo na 2 základních úrovních. První úroveň testovala funkčnost základní aplikace v prohlížečích jen se základním vybavením bez pokročilých webových technologií a CSS. Druhá úroveň testovala funkčnost HTML5 aplikace v moderních prohlížečích. Pro každý jednotlivý test se vytvoří nový adresář na KivFS systému, ve kterém proběhne test. Testování proběhlo ručně.

Základní aplikace

Pro testování základní aplikace jsem použil následující scénář:



Obrázek 4.1: Automatický test v prohlížeči Firefox 20.0.

1. Vytvoří se adresář *adresar* pomocí formuláře.
2. V adresáři 1 se vytvoří adresář *podadresar* pomocí formuláře.
3. Uploaduje se soubor do adresáře 1 pomocí formuláře.
4. Soubor se stáhne zpět pomocí kliknutí na odkaz souboru.
5. Soubor se přesune z adresáře 1 do adresáře 2 pomocí formuláře.
6. Soubor se stáhne zpět pomocí kliknutí na odkaz souboru.
7. Soubor se smaže pomocí formuláře.
8. Oba adresáře se smažou pomocí formuláře.

Aplikace byla funkční ve všech testovaných prohlížečích na různých platformách: Firefox 20.0, Google Chrome 26.0, Opera 12.12, Internet Explorer 10.0 a Android Browser (základní prohlížeč OS Android).

HTML5 aplikace

Pro testování HTML5 aplikace jsem použil následující scénář:

1. Vytvoří se adresář *adresar* pomocí tlačítka v panelu.
2. V adresáři 1 se vytvoří adresář *podadresar* pomocí tlačítka v panelu.
3. Uploaduje se soubor do adresáře 1 pomocí Drag & Dropu. Soubor text.txt má velikost 512bajtů náhodných textových dat.
4. Soubor se stáhne zpět pomocí dvojkliku a zkontroluje se shoda obsahu.
5. Soubor se přesune z adresáře 1 do adresáře 2.
6. Soubor se stáhne pomocí tlačítka.
7. Soubor se smaže pomocí tlačítka.
8. Oba adresáře se smažou pomocí tlačítka - napřed adresář 2, potom adresář 1.

Testovací scénář proběhl v následujích prohlížečích a platformách:

- Windows 8, 32 bit
  - Firefox 20.0.1
    100% funkčnost.
  - Google Chrome 26.0.1410.64
    100% funkčnost.
  - Opera 12.12
    100% funkčnost.
  - Internet Explorer 10.0.9200.16540
    100% funkčnost.
  - Internet Explorer 10.0.9200.16540, browser mode IE9
    V rozšířené verzi není možný upload a přesouvání. Prohlížeč nedostatečně podporuje Drag & Dropu.
- Ubuntu Linux 12.04 LTS

  - Firefox 20.0
    V rozšířené verzi není možný upload, prohlížeč nepodporuje upload pomocí Drag & Dropu.
  - Google Chrome 26.0.1410.63
    100% funkčnost.
  - Chromium 25.0.1364.160
    100% funkčnost.

- Windows Vista 32 bit

  - Firefox 18.0.2, 20.0.1
    100% funkčnost.
  - Google Chrome 26.0.1410.64
    100% funkčnost.
  - Opera 12.15
    100% funkčnost.
  - Internet Explorer 7.0.6001.18000
    Nefunkční - v aplikaci nefungují ani základní operace.

Kromě tohoto testování proběhlo testování bez scénáře zkoušením složitějších kombinací pro ověření možných problémů. Během testování se například zjistilo, že Drag & Drop funguje i mezi okny prohlížeče (Firefox, Chrome) a někdy i mezi okny různých prohlížečů (Firefox -> Chrome - opačný směr nejde kvůli bezpečnostní politice Firefoxu), což umožňuje používat klienta jako "dvoupanelový"manažer, podobně jako Total Commander nebo Servant Salamander pomocí dvou oken. Tato funkčnost má nevýhodu u zdrojového okna, kde se může stát, že není zaznamenána akce "drop", protože proběhne v druhém okně prohlížeče. K takovéto funkčnosti mezi prohlížeči jsem nenašel na w3c žádnou dokumentaci, která by tento jev více dokumentovala a až čas ukáže, jestli taková funkčnost bude v budoucích prohlížečích možná. Z bezpečnostního hlediska v tom není problém, protože informace, která se posílá mezi okny, je jen absolutní cesta k souboru a samotným aktem Drag & Drop z jednoho prohlížeče do druhého se dá předpokládat, že uživatel chce tyto informace druhému prohlížeči zpřístupnit podobně jako je to při uploadu pomocí Drag & Drop a zpřístupnění souboru v prohlížeči.

4.2.3 Chybové stavy

Chyba může nastat na několika úrovních, na všech úrovních je uživatel o chybě informován, ale kvalita informace o chybě se může lišit:

- Neočekávaná výjimka v aplikaci na serveru. Server odpoví chybovou stránkou, případně json objektem s informací o výjimce, které se klientovi zobrazí. Takovou chybou může být například nedostatečná autorizace.
- Chyba při komunikaci s KivFS. Server odpoví chybovou stránkou s konkrétní KivFS chybou, případně json objektem s informací o chybě, která se klientovi srozumitelně zobrazí. Takovou chybou může být například snaha o vytvoření existujícího adresáře.
- Kritická chyba na serveru. Server odpoví chybovou stránkou a kódem 5xx. Klientovi se zobrazí maximálně základní informace o chybě. Takovou chybou může být například nedostatek paměti na serveru.
- Klientská chyba. Chyba při běhu aplikace se stane chybou v prohlížeči bez komunikace se serverem. Toto může mít v krajním případě za následek neprovozuschopný stav klientské aplikace a nutnost restartu (obnovení okna v prohlížeči). Tento stav by neměl za normálních okolností nastat, protože aplikace je testována.

4.2.4 Měření

Měření Java knihovny měřilo komunikaci Java knihovny přímo s KivFS. Toto měření měří komunikaci webové aplikace s KivFS skrz proxy .

Pro měření jsem využil stejně jako pro testování základní funkčnosti framework qunit.js. Testy měření jsou naprogramovány jako automatické. Tímto způsobem se musí programově soubor velké velikosti uměle ”vytvořit”. V Javascriptu se vytvoří soubor pomocí objektu Blob a celý soubor je v paměti. Prohlížeče mají problém tak velký soubor udržovat v paměti, přestože reálný soubor při uploadu zpracují i mnohem větší bez problémů. Jako hranici, kterou prohlížeče zvládnou pro automatické testování, jsem stanovil 30MB, pro testování jsem použil hodnotu 20MB, protože to je i hodnota, kterou jsem použil při měření ve své bakalářské práci.

K měření je nutné poznamenat, že dokud se soubor neuploadne na HTTP server, nezačne se nahrávat na KivFS. Toto je vlastnost servletů. Kontejner nespustí obslužnou metodu, dokud nejsou dostupná všechna data. Toto by bylo možné obejít při použití jiné technologie na serveru. To znamená, že i při maximální rychlosti uploadu na server i nahrávání na KivFS je maximální teoretická rychlost poloviční, protože se musí nahrávat dvakrát stejně velký soubor sériově po sobě - jednou na HTTP server a poté na KivFS.

Při downloadu toto omezení neplatí. Je možné začít posílat data, jakmile přijde první odpověď z KivFS a posílat soubor průběžně i v případě zipu, protože zip archiv se vytváří sekvenčně. Tady je opět maximální teoretická rychlost rovná maximální rychlosti linky, i když se dá předpokládat, že režie na HTTP serveru rychlost značně sníží.

Při testování se nedá příliš měřit doba režie, protože nevíme, kdy se reálně posílají data a kdy se posílají řídící příkazy (pošle se najednou příkaz i data).

Testovací počítač byl virtuální stroj a měl následující parametry:

```
Procesory: Intel(R) Xeon(TM) CPU 2.14GHz
Operační paměť: 1GB
Operační systém: Windows 7 32bit
Souborový systém s testovacími daty: soubory jsou v paměti
Síťové připojení: 1Gbit
```

Upload a download 20MB souboru

měření	1	2	3	4	průměr
rychlost uploadu (MB/s)	3,9	3,9	4,4	4,5	4,2
rychlost downloadu (MB/s)	2,7	2,8	2,8	2,6	2,7
doba uploadu (s)	5,1	5,1	4,6	4,4	4,8
doba downloadu (s)	7,5	7,2	7,0	7,5	7,3
celková doba testu (s)	12,7	12,3	11,7	12,1	12,2

Tabulka 4.8: Výsledky měření uploadu a downloadu 20MB souboru v prohlížeči.

Když porovnám výsledky s bakalářskou prací, tak se výkon znatelně zvýšil - v bakalářské práci byla rychlost uploadu v průměru 353KB/s, což je více než

10x méně a downloadu 570KB/s, což je cca 5x méně. Zvětšení výkonu vzniká především díky použití nativních možností prohlížeče pro upload. V bakalářské práci jsem použil java Applet pro upload více souborů najednou, protože tou dobou ještě nebylo nativně možné. Dalším důvodem je posílání dat při downloadu rovnou klientovi. V bakalářské práci jsem napřed soubor stáhl z KivFS do dočasného adresáře, a pak ho začal posílat klientovi. Tato rychlost na druhou stranu je nižší než rychlost Java knihovny. Tady je nutné si uvědomit, že test běží v interpretovaném jazyce - Javascriptu a také oproti Java knihovně přibyla režie na proxy a HTTP protokolu.

Upload a download adresáře malých souborů

V tomto testu jsem uploadoval a následně downloadoval zpět 512 souborů o velikosti 128KB, stejně jako ve své bakalářské práci.

měření	1	2	3	4	průměr
rychlost uploadu (MB/s)	1,5	1,4	1,4	1,5	1,4
rychlost downloadu (MB/s)	1,3	1,3	1,3	1,8	1,4
doba uploadu (s)	43,9	46,9	46,7	44,0	45,4
doba downloadu (s)	48,9	49,6	50,1	37,1	46,4
celková doba testu (s)	103,2	101,2	99,7	80,5	96

Tabulka 4.9: Výsledky měření uploadu a downloadu 512 128KB souborů v prohlížeči.

Přenos souborů je opět znatelně rychlejší, než v bakalářské práci, kde byla rychlost uploadu průměrně 188KB/s a downloadu průměrně 340KB/s.

Upload a download adresářové struktury s malými soubory

V tomto testu jsem uploadoval a následně downloadoval zpět struktur 5 adresářů, každý obsahující 512 souborů o velikosti 128KB, stejně jako ve své bakalářské práci.

Naměřené hodnoty jsou srovnatelné s hodnotami předchozího testu.

měření	1	2	3	4	průměr
rychlost uploadu (MB/s)	1,4	1,4	1,5	1,4	1,4
rychlost downloadu (MB/s)	1,3	1,3	1,4	1354	1,3
doba uploadu (s)	225,5	226,4	223,2	230,9	226,6
doba downloadu (s)	224,2	247,4	237,3	247,8	244,4
celková doba testu (s)	526,1	561,2	604,0	550,5	560,4

Tabulka 4.10: Výsledky měření uploadu a downloadu 512 128KB souborů v prohlížeči.

5 Závěr

Hlavním cílem práce bylo vytvoření webového klienta. Tento úkol byl splněn. Implementace byla rozdělena na několik částí. Byla vytvořena proxy jako alternativní vstupní bod do KivFS. Dále byl navržen protokol nad HTTP, kterým je možné komunikovat s KivFS prostřednictvím alternativního vstupního bodu. Nový vstupní bod do KivFS otevřel prostor pro klienty do dalších platforem, které nepodporují TCP nebo Kerberos. Tato proxy a tento protokol používají nově vytvoření www klienti - jednoduchý HTML klient, který funguje v každém z testovaných prohlížečů a klient v HTML5, který má pokročilé funkce a je uživatelsky přívětivější. Pro ověření funkčnosti vytvořených komponent byly vytvořeny automatické testy, které testují proxy, protokol a zčásti i klienta. Částečně byl klient otestován i ručně, obzvlášť grafické rozhraní. Oproti mé bakalářské práci došlo v některých případech až k desetinásobnému zrychlení. Naproti tomu testy pro velké množství malých souborů jsou méně výkonnější z důvodu vyšší režie na serveru a zesložitění protokolu. Přesto jsou ve výkonu rezervy a proxy i www klient se může dále zrychlovat. Jazyk HTML5 se rychle rozvíjí a nabízí čím dál více technologií, které by klienta zkvalitnily a do budoucna jistě přibudou další. Díky navrženému protokolu je možné vytvořit klienty i v jiných prostředích, které podporují HTTP protokol a nepodporují technologie nutné pro standardní spojení s KivFS.

Literatura

[Armstrong(1997)] ARMSTRONG, T. *Designing and Using ActiveX Controls*. John Wiley and Son, 1997. ISBN 1558515038.

[Bill Evjen(2008)] BILL EVJEN, D. R. S. H. *Professional ASP.NET 3.5: In C# and VB*. : Wrox, 2008. ISBN 0470187573.

[Bogus(2008)] BOGUS, A. *Lighttpd*. Packt Publishing, 2008. ISBN 1847192106.

[Braam(2012)] BRAAM, P. J. *The Coda Distributed File System*. Addison-Wesley Professional, 2012. ISBN online.

[Callaghan(2000)] CALLAGHAN, B. *NFS Illustrated*. Addison-Wesley Professional, 2000. ISBN 0201325705.

[Campbell(1998)] CAMPBELL, R. *Managing AFS: The Andrew File System*. Prentice Hall, 1998. ISBN 0138027293.

[Carpenter(2012)] CARPENTER, T. *Microsoft Windows Operating System Essentials*. Sybex, 2012. ISBN 1118195523.

[Christopher Negus(2008)] CHRISTOPHER NEGUS, F. C. *BSD UNIX Toolbox: 1000+ Commands for FreeBSD, OpenBSD and NetBSD*. Wiley, 2008. ISBN 0470376031.

[David P. Bovet(2005)] DAVID P. BOVET, M. C. P. D. *Understanding the Linux Kernel, Third Edition*. O'Reilly Media, 2005. ISBN 0596005652.

[Engelschall(2000)] ENGELSCHALL, R. S. *Apache Desktop Reference*. : Pearson Education, 2000. ISBN 0201604701.

[Evi Nemeth(2010)] EVI NEMETH, T. R. H. B. W. G. S. *UNIX and Linux System Administration Handbook*. Prentice Hall, 2010. ISBN 0131480057.

[Fazekaš(2010)] FAZEKAš, J. *KIVFS - Klient pro GNU/Linux s pomocí jaderného modulu [bakalářská práce].* Západočeská univerzita, Fakulta aplikovaných věd, 2010. ISBN 000157101.

[Foust(2010)] FOUST, B. *BlackBerry Java Application Development: Beginner's Guide.* Packt Publishing, 2010. ISBN 1849690200.

[Hartl(2012)] HARTL, M. *Ruby on Rails Tutorial: Learn Web Development with Rails (2nd Edition).* Addison-Wesley Professional, 2012. ISBN 0321832051.

[Haseman(2008)] HASEMAN, C. *Android Essentials.* Apress, 2008. ISBN 1430210648.

[Hovorka(2010)] HOVORKA, K. *KIVFS - Webový klient [bakalářská práce].* Západočeská univerzita, Fakulta aplikovaných věd, 2010. ISBN 000157129.

[Jaroš(2012)] JAROš, P. *KIVFS - klient pro GNU/LINUX s použitím FUSE [diplomová práce].* Západočeská univerzita, Fakulta aplikovaných věd, 2012. ISBN 000180517.

[Jason Brittain(2003)] JASON BRITTAIN, I. F. D. *Tomcat: The Definitive Guide.* O'Reilly Media, 2003. ISBN 0596003188.

[Javid Jamae(2009)] JAVID JAMAE, P. J. *JBoss in Action: Configuring the JBoss Application Server.* Manning Publications, 2009. ISBN 1933988029.

[K. C. Hopson(1996)] K. C. HOPSON, S. E. I. *Developing Professional Java Applets.* Sams Publishing, 1996. ISBN 1575210835.

[Kevin Mukhar(2006)] KEVIN MUKHAR, J. L. W. J. C. C. Z. *Beginning Java EE 5: From Novice to Professional.* : Apress, 2006. ISBN 1590594703.

[Kou(2010)] KOU, X. *GlassFish Administration.* Packt Publishing, 2010. ISBN 1847196500.

[Lurig(2008)] LURIG, M. *PHP Reference: Beginner to Intermediate PHP5.* : Lulu.com, 2008. ISBN 143571590X.

[MacCaw(2012)] MACCAW, A. *The Little Book on CoffeeScript.* O'Reilly Media, 2012. ISBN 1449321054.

[Mike Cantelon(2013)] MIKE CANTELON, N. R. T. H. *Node.js in Action*. Manning Publications, 2013. ISBN 1617290572.

[Mike Volodarsky(2008)] MIKE VOLODARSKY, B. H. B. C. S. S. C. A. M. K. M. M. I. T. O. L. *Internet Information Services (IIS) 7.0 Resource Kit*. Microsoft Press, 2008. ISBN 8120335120.

[Nedelcu(2010)] NEDELCU, C. *Nginx HTTP Server*. Packt Publishing, 2010. ISBN 1849510865.

[Pinto(2007)] PINTO, M. *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Wiley, 2007. ISBN B000SFC7S0.

[Pogue(2012)] POGUE, D. *OS X Mountain Lion: The Missing Manual*. O'Reilly Media, 2012. ISBN 1449330274.

[Rama Turaga(2006)] RAMA TURAGA, P. V. S. O. C. *WebSphere Application Server: Step by Step*. Mc Press, 2006. ISBN 1583470611.

[Rescorla(2000)] RESCORLA, E. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley Professional, 2000. ISBN 0201615983.

[Roden(2010)] RODEN, T. *Building the Realtime User Experience: Creating Immersive and Interactive Websites*. O'Reilly Media, 2010. ISBN 0596806159.

[Sadun(2012)] SADUN, E. *The Core iOS 6 Developer's Cookbook*. Addison-Wesley Professional, 2012. ISBN 0321884213.

[Sales(2005)] SALES, J. *Symbian OS Internals: Real-time Kernel Programming*. Wiley, 2005. ISBN 0470025247.

[Sanderson(2012)] SANDERSON, D. *Programming Google App Engine*. Google Press, 2012. ISBN 144939826X.

[Team(2012)] TEAM, A. C. *Adobe Flash Professional CS6 Classroom in a Book*. Adobe Press, 2012. ISBN 032182251X.

[Tung(1999)] TUNG, B. *Kerberos: A Network Authentication System*. Addison-Wesley Professional, 1999. ISBN 0201379244.

[Vincent Massol(2003)] VINCENT MASSOL, T. H. *JUnit in Action*. Manning Publications, 2003. ISBN 1930110995.

[Watters(2005)] WATTERS, P. *Solaris 10 The Complete Reference*. McGraw-Hill Osborne Media, 2005. ISBN 0072229985.

A Uživatelský manuál

Tento manuál popisuje ovládání webových aplikací pro KivFS. Obsahuje ovládání serverové verze, která téměř neobsahuje Javascript a ovládání HTML5 verze, která naopak požaduje moderní internetový prohlížeč.

Manuál je rozdělený na dvě části pro každou tuto aplikaci a popisuje, jak v každé aplikaci udělat všechny možné operace.

A.1 Úvodní obrazovka

Úvodní obrazovka obsahuje formuláře pro KivFS operace na serveru, odkaz na prohlížení souborů a odkaz na HTML5 aplikaci.

A.2 HTML5 aplikace

Spuštění HTML5 aplikace probíhá navštívením souboru app.html v kořenovém adresáři webového projektu (pokud se spouští lokálně), nebo adresy, kde aplikace běží. Při spuštění se objeví formulář pro připojení. V horní části aplikace jsou ovládací tlačítka pro většinu operací, aby byly dostatečně na očích.

A.2.1 Připojení

Připojení probíhá formulářem, který se objeví hned při spuštění aplikace. Pro připojení je nutné vyplnit všechny přihlašovací údaje:

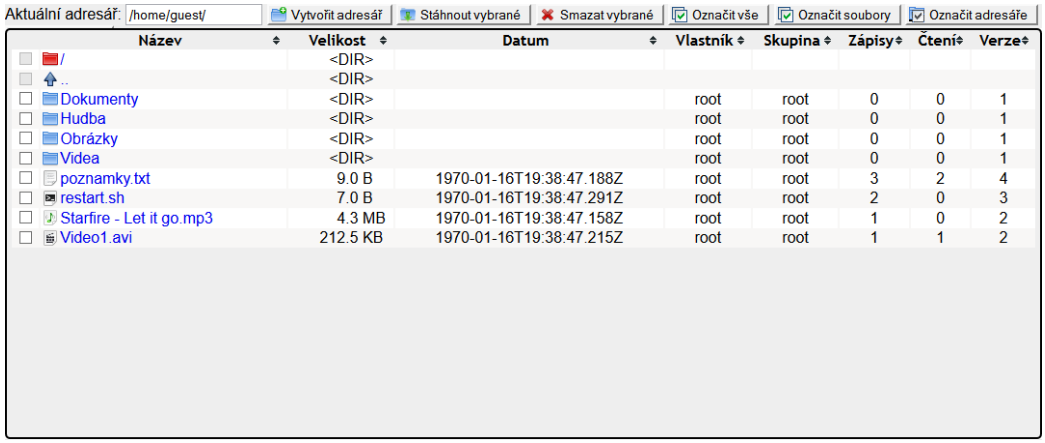
- Adresa http proxy serveru
  Tato hodnota musí obsahovat adresu na proxy server.
- Hostname KivFS serveru
  Tato hodnota musí obsahovat hostname nebo ip adresu KivFS serveru.

- Port
  Tato hodnota musí obsahovat port, na kterém KivFS běží.
- Login
  Tato hodnota musí obsahovat přihlašovací jméno v KivFS.
- Heslo
  Tato hodnota musí obsahovat přihlašovací heslo do KivFS.
- Úvodní adresář
  Tato hodnota obsahuje úvodní adresář, který se vypíše po připojení.

Tlačítkem *Přihlásit* dojde k připojení na proxy, ověření proti Kerberu, připojení na KivFS, ověření proti KivFS a následně k vypsání úvodního adresáře. Pokud kdykoliv během procesu přihlašování dojde k chybě, uživatel je o chybě informován oznamovacím oknem.

A.2.2 Procházení adresářů

Pro změnu adresáře stačí na adresář dvojkliknout. Pro změnu o úroveň výše je nutné kliknout na adresář s název ..(dvě tečky), který reprezentuje adresář o úroveň výš. Je také možné použít v horním panelu textové pole s našeptávací nápovědou, kam se dá zadat cesta k adresáři ručně.



Obrázek A.1: Prohlížení adresáře v prohlížeči Firefox 20.0.

A.2.3 Vytváření adresáře

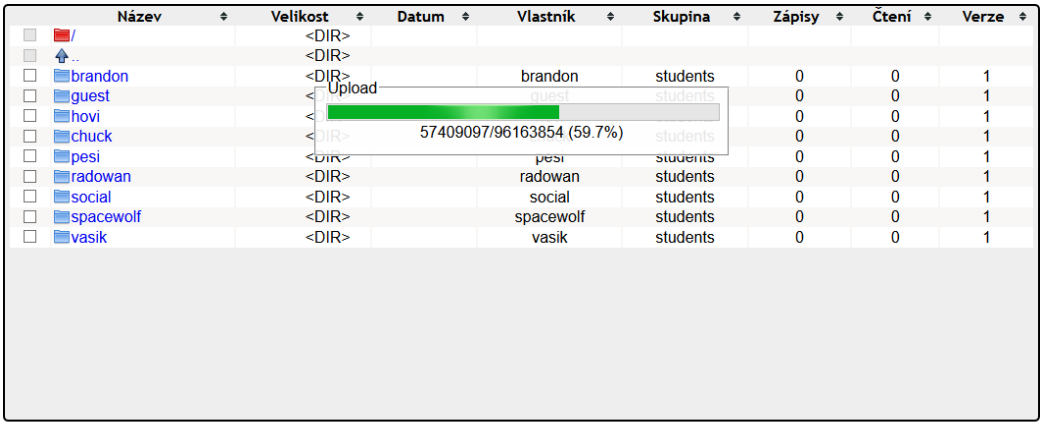
Pro vytvoření adresáře slouží tlačítko *Vytvořit adresáře* v horním panelu aplikace. Po zmáknutí tlačítka se objeví dotazovací dialog na jméno adresáře. Adresář se vytvoří v aktuálním prohlíženém adresáři.

A.2.4 Mazání

Pro smazání adresáře nebo souboru je nutné adresář nebo soubor nejprve označit checkboxem, který je na řádce vlevo. Jakmile je adresář nebo adresář označený, je možné použít tlačítko *Smazat označené*, které provede smazání.

A.2.5 Upload souboru

Upload souboru probíhá pouhým Drag & Dropem. Stačí vzít soubor z jeho složky na počítači a přetáhnout ho do okna prohlížeče. Aplikace začne okamžitě uploadovat a ukazovat průběh zobrazeným formulářem. Jakmile formulář zmizí, soubor je uploadovaný.



Obrázek A.2: Prohlížení upload souboru v prohlížeči Firefox 20.0.

A.2.6 Stáhnutí souboru

Stáhnutí souboru probíhá pouhým dvojklikem na soubor. Pro stahování více souborů nebo adresářů najednou je možné soubory a adresáře označit a použít tlačítko *Stáhnout vybrané*, které provede stáhnutí všech vybraných souborů a adresářů jako 1 soubor v archivu zip.

A.2.7 Přesunutí

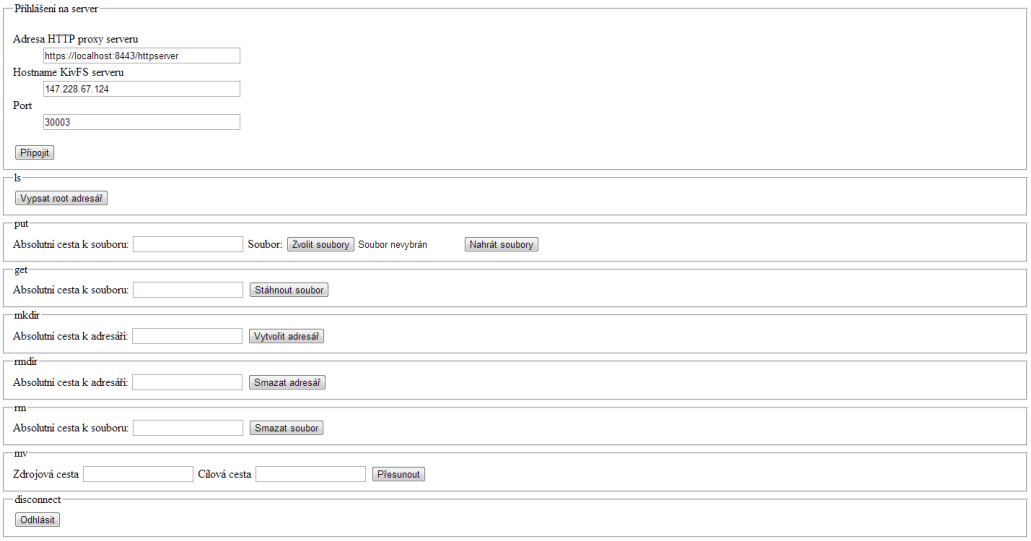
Přesouvání je realizováno Drag & Dropem. Pro přesunutí o úroveň výše je možné použít adresář s názvem ... Na jedné obrazovce je možné přesunout jen do adresáře o úroveň výš, do kořenového adresáře / nebo do podadresářů. V případě nutnosti přesunutí do jiného adresáře je možné aplikaci otevřít ve dvou oknech a realizovat Drag & Drop mezi okny.

A.2.8 Odpojení

Pro odpojení stačí použít tlačítko *Odpojit* v horním panelu aplikace. Odpojení provede kompletní odhlášení a veškeré další požadavky na server budou neplatné do dalšího přihlášení.

A.3 Serverová aplikace

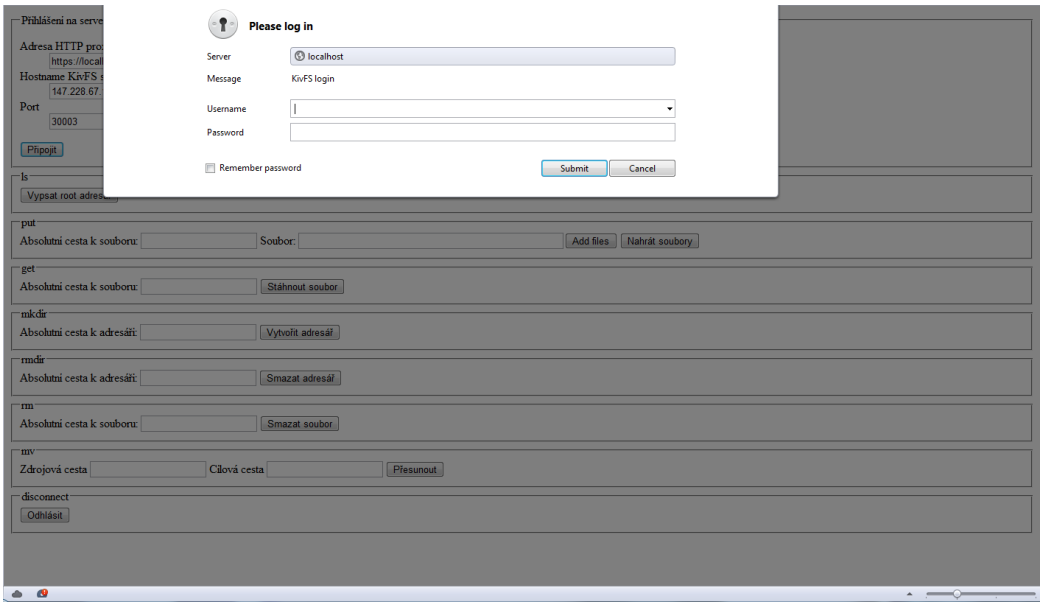
Serverová aplikace má méně komfortní ovládání, ale je funkční prakticky v každém existujícím prohlížeči včetně textových. Místo panelu s tlačítky nahoře v HTML5 aplikaci je odkaz *Další příkazy*. Důvod, proč jsou další příkazy na zvláštní obrazovce, je jejich velikost na obrazovce. Každý příkaz se provádí HTML formulářem a pokud by měly být všechny na stejné obrazovce, aplikace by ztratila na přehlednosti.



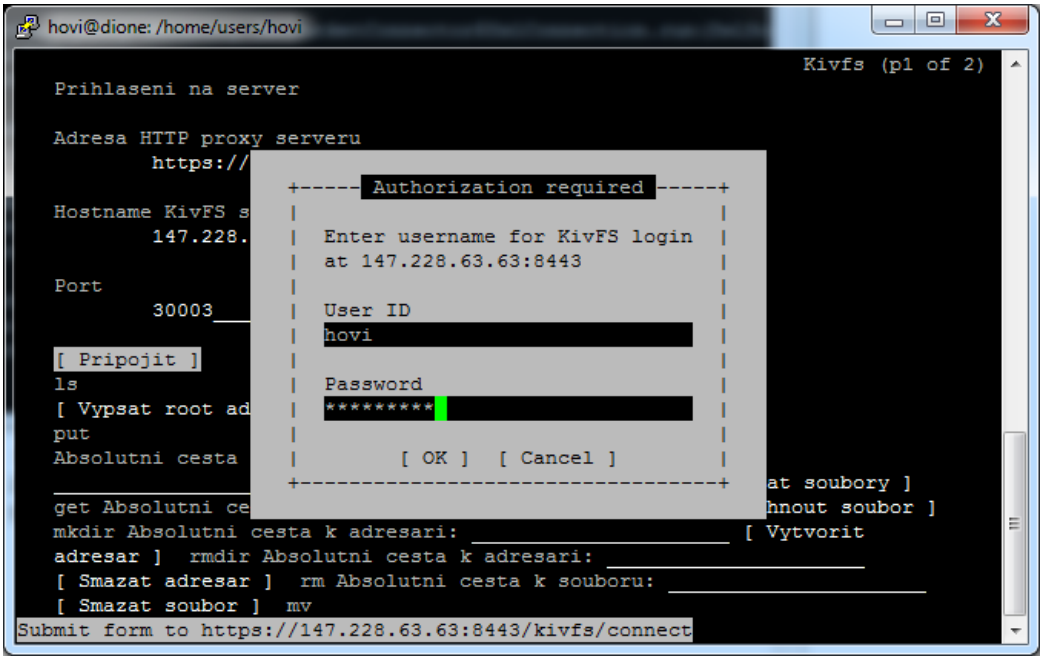
Obrázek A.3: Ovládácí formuláře serverové aplikace.

A.3.1 Přihlášení

Přihlášení proběhne vyplněním podobného formuláře jako u HTML5 aplikace, chybí položka *Úvodní adresář*, protože po přihlášení nenásleduje obrazovka výpisu adresáře. Dále chybí ve formuláři přihlašovací údaje, protože na ně se zeptá sám prohlížeč hned poté (HTTP Autentizace).



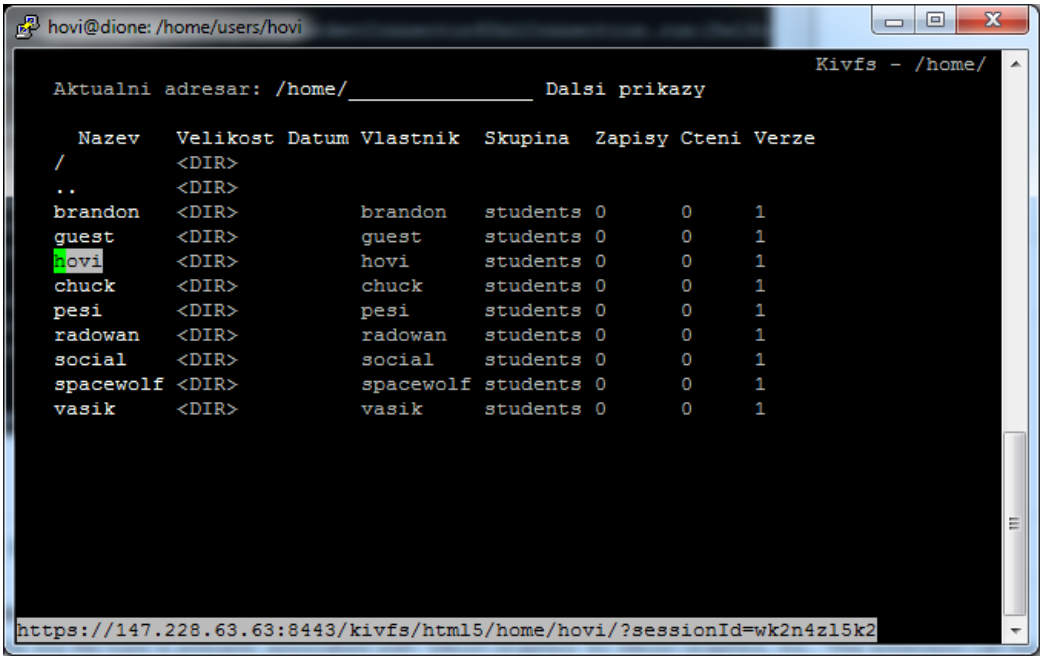
Obrázek A.4: Zadávání uživatelského jména a hesla v prohlížeči Opera.



Obrázek A.5: Zadávání uživatelského jména a hesla v prohlížeči links.

A.3.2 Procházení adresářů

Pro změnu adresáře stačí kliknout na odkaz s názvem adresáře.



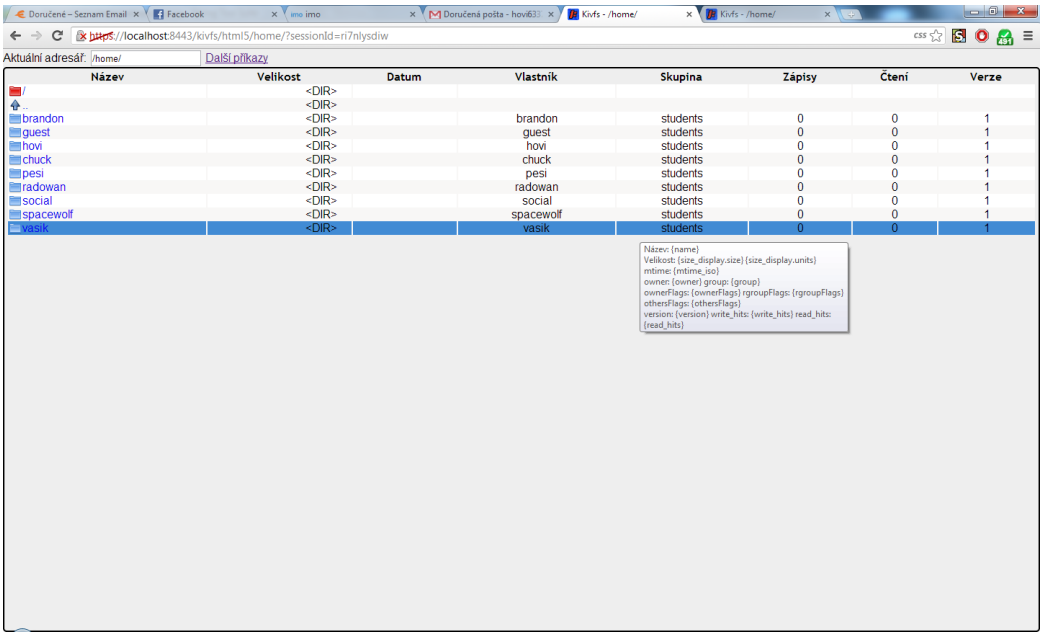
Obrázek A.6: Výpis adresáře v textovém prohlížeči links.

A.3.3 Vytvoření adresáře

Pro vytvoření adresáře je nutné nejprve kliknout na odkaz *Další příkazy*, který přesměruje na formuláře. Ve formuláři pro vytvoření adresáře je nutné zadat jméno adresáře včetně absolutní cesty a použít tlačítko *Vytvořit adresář*.

A.3.4 Mazání

Pro smazání adresáře nebo souboru je nutné nejprve kliknout na odkaz *Další příkazy*, který přesměruje na formuláře. Ve formuláři pro mazání je nutné zadat jméno souboru nebo adresáře včetně absolutní cesty a použít tlačítko



Obrázek A.7: Prohlížení adresáře serverové aplikace v prohlížeči Firefox 20.0.

Smazat soubor nebo *Smazat adresář*. Mazání adresáře a mazání souboru mají 2 různé formuláře.

A.3.5 Upload souboru

Pro smazání adresáře nebo souboru je nutné nejprve kliknout na odkaz *Další příkazy*, který přesměruje na formuláře. Ve formuláři je zadat soubory, které chcete uploadovat a absolutní cestu adresáře na serveru, kam se mají adresáře nahrát.

A.3.6 Stáhnutí souboru

Pro stáhnutí souboru stačí kliknout na odkaz s názvem souboru.

A.3.7 Přesunutí

Pro přesunutí adresáře nebo souboru je nutné nejprve kliknout na odkaz *Další příkazy*, který přesměruje na formuláře. Ve formuláři je nutné zadat původní absolutní cestu zdrojového souboru nebo adresáře a novou absolutní cestu, kam má být soubor nebo adresář přesunut.

Odpojení

Pro odpojení stačí použít odkaz *Další příkazy* a v dalších příkazech použít odkaz *Odpojit*.