VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

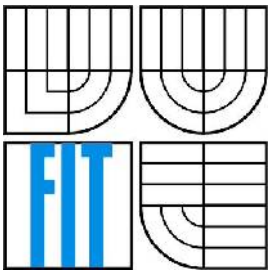
FTP SERVER PRO WINDOWS MOBILE 6

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUKÁŠ ČERNÝ
AUTHOR

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

FTP SERVER PRO WINDOWS MOBILE 6

FTP SERVER FOR WINDOWS MOBILE 6

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ ČERNÝ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. PETR ČÁSTEK

BRNO 2009

Abstrakt

Tato práce se zabývá postupně protokolem FTP, jeho využitím, příkazy a komunikací mezi klientem a serverem. Je probrána platforma Windows Mobile, běhové prostředí .NET Compact Framework, jazyk C# a programování v něm. Dále práce dokumentuje návrh, implementaci a testování FTP serveru pro mobilní zařízení.

Abstract

This work deals with FTP protocol, its using, commands and communication between client and server site. It discusses Windows Mobile platform, .NET Compact Framework runtime environment, C# language and programming with it. The work documents concept, implementation and testing of FTP server for mobile devices.

Klíčová slova

FTP, PDA, Windows Mobile, jazyk C#, .NET Compact Framework

Keywords

FTP, PDA, Windows Mobile, C# language, .NET Compact Framework

Citace

Lukáš Černý: FTP server pro Windows Mobile 6, bakalářská práce, Brno, FIT VUT v Brně, 2009

FTP server pro Windows Mobile 6

Prohlášení

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

……
Lukáš Černý
20.5.2009

Poděkování

Děkuji Ing. Petru Částkovi za odborné vedení a návrhy pro vylepšení jak aplikace tak i technické zprávy.

Obsah

1 Úvod

Mobilní zařízení s operačním systémem Windows Mobile již poslední dobou nejsou, díky čím dál častější reklamě v médiích a hlavně příznivější ceně, pouze záležitostí vysoce postavených manažerů a podnikatelů, ale také normálních lidí. Díky obrovskému množství různých aplikací pro tuto platformu je možné zařízení programově vybavit tak, že jeho použití nekončí u adresáře či seznamu úkolů. Existují například implementace SSH klientů, HTTP serveru, FTP klientů a spousty dalších.

Třeba takový protokol FTP je v současnosti jedním z nejrozšířenějších protokolů pro přenos souborů. Vzhledem k jeho nezávislosti na použité platformě je možné ho využívat téměř kdekoliv. Avšak pro dnes stále více se rozšiřující platformu Windows Mobile je použitelných řešení pro využití serveru FTP žalostně málo. Možná je to způsobeno nedůvěrou v dostatečný výkon mobilních zařízení, ale pokud se podíváme na parametry například PDA (Personal Digital Assistant) s 514 MHz procesorem, 64 MB operační paměti a několika gigabajtovým úložištěm, dostaneme se daleko za možnosti počítačů z doby kdy byl protokol FTP navržen. Já osobně bych, jako vlastník jednoho takového zařízení, uvítal možnost provozovat na něm FTP server, například pro jednoduchý přenos fotografií ze zařízení bez nutnosti ho připojovat k počítači.

Celá aplikace je implementována v jazyce C#. Pro vývoj bylo použito vývojové prostředí Microsoft Visual Studio 2005 s doinstalovaným balíkem Windows Mobile 6 SDK pro vývoj mobilních aplikací. Výsledná aplikace je navržena pro běhové prostředí .NET Compact Framework 2.0, ale díky zpětné kompatibilitě prostředí je možné ji bez problému provozovat i pod novějšími verzemi.

Druhá kapitola se zabývá protokolem FTP. Je v ní vysvětlen způsob komunikace mezi klientem a serverem. Nastíněno k čemu slouží jaké příkazy a jaké odpovědi používá server. Dále jsou zmíněny způsoby zabezpečení protokolu FTP.

Třetí kapitola se věnuje základním informacím o platformě Windows Mobile, běhovému prostředí .NET Compact Framework a programování za pomocí jazyku C#.

Čtvrtá kapitola se věnuje samotnému návrhu a implementaci FTP serveru. Je v ní vysvětleno rozdělení do jednotlivých projektů a způsob implementace jednotlivých logických celků aplikace.

V páté kapitole jsou informace o testování výsledné aplikace a použitých klientech.

2 Protokol FTP

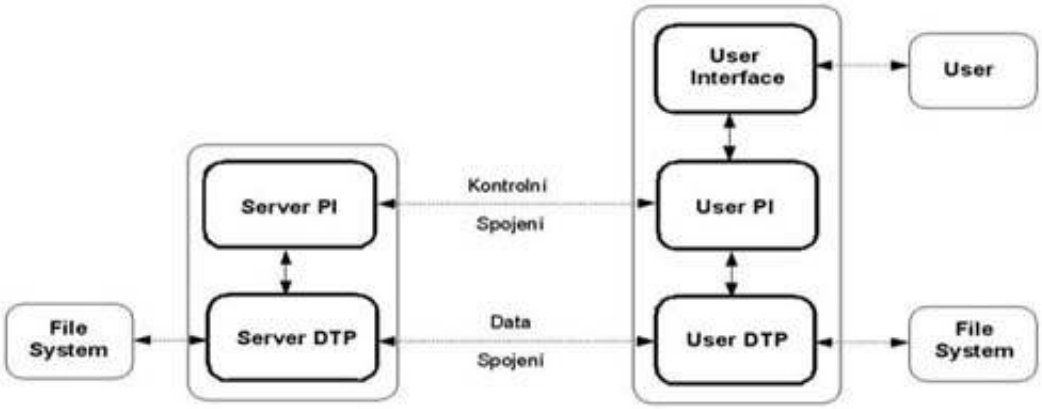
Protokol FTP (File Transfer Protokol) je protokolem aplikační vrstvy z rodiny TCP/IP sloužící, jak již jeho název napovídá, k přenosu souborů. Díky tomu, že se jedná o protokol aplikační vrstvy, je možné ho provozovat v podstatě na libovolném operačním systému (platformní nezávislost). Jedná se o poměrně starý protokol, je definován normou RFC 959 [1] vydanou v říjnu 1985, i přesto je však i dnes velmi rozšířený. Protokol využívá model klient-server. FTP server poskytuje přístup k datům pro ostatní aplikace. Klient se k serveru připojí a pomocí předem definované sady příkazů může provádět různé operace např. stažení souboru ze serveru, uložení souboru na server, výpis souborů apod. Informace o protokolu FTP jsem čerpal z [1, 5, 11], informace o zabezpečených variantách jsou čerpány z [4, 5, 9, 10].

2.1 Kontrolní a datové spojení

Standardně server naslouchá na TCP portu 21 na příchozí spojení od klienta. Toto navázané spojení se označuje jako spojení kontrolní (někdy také řídicí) a slouží k přenosu příkazů od klienta k serveru a odpovědí serveru zpět klientovi. Kontrolní spojení je navázáno po celou dobu komunikace serveru a klienta.

Oproti tomu datové spojení slouží k přenosu dat mezi serverem a klientem a navazuje se většinou před samotným přenosem na předem domluveném TCP portu. Tedy nemusí být aktivní po celou dobu komunikace. Protokol FTP definuje dvě možnosti navázání datového spojení. První možností je spojení aktivní, kdy se server připojí ke klientovi na předem dohodnutý port (klient tedy na dohodnutém portu naslouchá). Při tomto způsobu mohou ovšem nastat potíže, pokud se klient připojuje z privátní sítě a jeho adresa je překládána pomocí NAT (většina dnešních routerů již tento problém řeší). Druhý způsob je spojení pasivní, kdy na předem dohodnutém portu naslouchá server a klient se na něj připojí. U tohoto způsobu se již nevyskytuje výše zmíněný problém s překladem adres.

Schéma použití datového a kontrolního spojení je zobrazeno na obrázku 2.1 1. Server a user PI reprezentuje interpret příkazů protokolu. DTP (Data Transfer Process) reprezentuje proces zajišťující navázání a udržování datového spojení a přenos dat přes něj.



Obrázek 2.1 1: Schéma protokolu FTP zobrazující datové a kontrolní spojení

2.2 Způsoby přenosu a reprezentace dat

Z důvodů možné různé reprezentace dat u klienta a serveru protokol FTP definuje několik způsobů reprezentace použitých pro přenos a ukládání dat. Tyto způsoby jsou ASCII, IMAGE, LOCAL a EBCDIC:

- ASCII se používá převážně pro přenos textových souborů. Je tedy před přenosem třeba převést data z vnitřní reprezentace serveru na 8-mi bitovou ASCII reprezentaci.
- IMAGE slouží hlavně k přenosu binárních dat, data jsou odesílána jako posloupnost bitů.
- LOCAL využívají systémy, které nejsou bajtově orientované. Data se tedy převedou na danou bitovou délku, např. 36-ti bitové slovo.
- EBCDIC určuje, že data jsou v rozšířeném binárně kódovaném desítkovém výměnném kódu.

Poslední dva způsoby (LOCAL, EBCDIC) reprezentace dat se v dnešní době již převážně nepoužívají.

Dále je možné stanovit způsob přenosu dat mezi klientem a serverem. Protokol stanovuje tři možné způsoby a to:

- STREAM MODE, data jsou odesílána klientovi jako proud bajtů. Není třeba žádné další úpravy dat. Jako signalizace odeslání všech dat (např. souboru) je datové spojení uzavřeno.
- BLOCK MODE, data jsou odesílána jako série datových bloků předcházených tříbajtovou hlavičkou. Hlavička obsahuje dvě položky. První je popisovač bloku, který definuje např. zda jde o poslední blok, nebo restartovací značku. Druhá je informace o velikosti bloku.
- COMPRESSED MODE, je používán pro přenos dat v komprimované podobě.

V dnešní době se však používá pouze STREAM MODE.

2.3 Příkazy protokolu FTP

Příkazy protokolu jsou přenášeny po řídicím spojení. Příkazy i zprávy se přenáší v takzvaném telnetovém formátu, tedy jednotlivé příkazy (zprávy) jsou oddělené posloupností znaků <CRLF>.

Příkazy jsou posílány klientem serveru. Formát příkazu je následující:

```
PŘÍKAZ [PARAMETR1] [PARAMETR2]<CRLF>
```

Příkaz určuje zda jsou parametry povinné nebo ne. Například příkaz NOOP (nedělej nic) nemá parametr žádný, příkaz USER (přihlášení uživatele) má parametr jeden a příkaz TYPE (datová reprezentace) s prvním parametrem L má ještě doplňující parametr udávající bitovou délku.

V následujících podkapitolách nastíním funkci a použití některých z nich.

2.3.1 Příkazy PORT a PASV

Tyto příkazy slouží k ustanovení typu datového spojení. Příkaz PORT se využívá pro navázaní aktivního spojení, jeho parametrem je IP adresa a číslo portu, na které se má server připojit. Ve chvíli kdy klient odesílá příkaz serveru by měl již naslouchat na zadaném portu.

Oproti tomu příkaz PASV slouží k navázání spojení pasivního. Klient pošle serveru příkaz, ten otevře port a v odpovědi klientovi zašle danou IP adresu a číslo portu, na kterém naslouchá.

2.3.2 Příkazy USER, PASS, REIN a QUIT

Tyto příkazy slouží k autentizaci uživatele na serveru. Příkaz USER má jako jediný parametr přihlašovací jméno klienta. Většinou je klientem odeslán ihned po připojení k serveru. Server následně může klientovi v závislosti na svém nastavení poslat zprávu o úspěšném přihlášení nebo požadavek na heslo. Heslo klient serveru odesílá jako parametr příkazu PASS, který musí být (v případě, že je heslo vyžadováno) odeslán ihned po příkazu USER. Některé servery nemusí autentizaci vyžadovat a povolují anonymní přístup.

Příkaz REIN slouží k reinicializaci komunikace, tedy po jeho užití server odhlásí uživatele, nastaví výchozí cesty a chová se jako by se klient právě připojil. Kontrolní spojení zůstává nadále navázáno.

Příkaz QUIT slouží k odhlášení uživatele a ukončení komunikace. Pokud však probíhá přenos po datovém spojení tak server dokončí tento přenos a komunikaci ukončí až následně.

2.3.3 Příkazy CWD, PWD, LIST a NLST

Tato skupina příkazů se využívá k pohybu souborovým systémem FTP serveru. Příkaz `CWD` slouží k změně pracovního adresáře. Jeho jediným parametrem je cesta specifikující cílový adresář. Příkaz `PWD` slouží k vypsání cesty aktuálního pracovního adresáře a žádné parametry nevyžaduje.

Příkazy `LIST` a `NLST` slouží k výpisu obsahu pracovního adresáře, či adresáře zadaného parametrem. Obsah adresáře je klientovi odeslán po datovém spojení v kódování ASCII. Norma neurčuje formát odesílaných dat, je tedy možné (i když ne velmi rozumné) implementovat formát vlastní. Obecně uznávaným formátem, který podporuje většina FTP klientů je formát */bin/ls* tedy formát vracený unixovým příkazem *ls*. Příkaz `LIST` slouží k získání seznamu s detailnějšími informacemi, jako např. práva, čas poslední modifikace apod. Kdežto příkaz `NLST` vrací pouze seznam položek v daném adresáři.

Příklad záznamu v */bin/ls* formátu:

```
drwxr-xr-x 1 owner group            0 Apr 13 13:15 Directory
-rw-r--r-- 1 owner group         1383 Apr 10 11:15 ip.c
-rw-r--r-- 1 owner group          213 Aug 26 16:31 README
```

2.3.4 Příkazy STOR, RETR, APPE a DELE

Tyto příkazy slouží k práci se soubory. Příkaz `STOR` se používá k uložení souboru na server. Název souboru je zadán parametrem, pokud již soubor existuje je přepsán, jinak se vytvoří nový. Soubor se posílá po datovém spojení.

Příkaz `RETR` slouží k získání souboru ze serveru. Po přijetí příkazu server kopii souboru zadaného parametrem vystaví na datové spojení. Tato operace nijak neovlivní soubor umístěný na serveru.

Příkaz `APPE` slouží k připojení obsahu souboru vystaveného na datovém spojení k souboru již existujícímu na serveru. Jméno souboru je opět zadáno parametrem příkazu. Pokud daný soubor neexistuje, tak je vytvořen.

Příkaz `DELE` slouží k smazání souboru ze serveru. Pokud je třeba zabezpečení typu „Opravdu chcete daný soubor odstranit?“ je implementace záležitostí klienta.

2.3.5 Příkazy MKD a RMD

Příkazy slouží k práci s adresáři. Příkaz `MKD` je používán k vytvoření nového adresáře jehož cesta je zadána parametrem. Oproti tomu příkaz `RMD` slouží k odstranění adresáře zadaného parametrem. Případný dotaz na potvrzení provedení akce je stejně jako v případě příkazu `DELE` záležitostí implementovanou na klientovi.

2.3.6 Příkazy RNFR a RNTO

Příkazy `RNFR` a `RNTO` slouží k přejmenování souboru na FTP serveru. Jako první klient odešle serveru příkaz `RNFR` s parametrem určujícím který soubor (adresář) má být přejmenován. Následně klient odešle serveru příkaz `RNTO` s parametrem určujícím nový název daného souboru (adresáře). Server přejmenování provede pouze po úspěšném obdržení obou těchto příkazů ve správném pořadí (a po kontrole existence objektu a práv uživatele k němu).

2.3.7 Příkazy ALLO, MODE, TYPE a STRU

Příkaz ALLO slouží k alokaci prostoru na serveru, vzhledem k velikostem paměťového prostoru na dnešních serverech však víceméně postrádá smysl. Dnešní servery na něj odpovídají zprávou s kódem 202, který klientovi říká, že server příkaz neimplementuje, ale klient se má chovat jako kdyby úspěšně proběhl.

Příkazy TYPE a MODE slouží k nastavení způsobu přenosu a reprezentace dat[1]. TYPE slouží k nastavení způsobu reprezentace, dnešní servery implementují většinou pouze podporu ASCII a binárního (IMAGE) způsobu. MODE slouží k nastavení přenosu dat, dnes je obecně podporován pouze STREAM přenos.

Příkaz STRU slouží k nastavení struktury ukládaných dat na straně serveru. Je možné zvolit ukládání například formou souboru nebo záznamu. Dnes je však podporována pouze struktura FILE.

2.3.8 Příkazy HELP a NOOP

Příkaz `HELP` slouží k vyžádání užitečných informací od serveru, jedná se v podstatě o nápovědu pro uživatele využívající například klienty příkazové řádky. Server na tento příkaz odpovídá např. seznamem implementovaných příkazů, odpověď je odesílána klientovi zpět po kontrolním spojení. Příkaz může přijmout jako parametr název příkazu pro nějž má být klientovi vrácena nápověda.

Příkaz `NOOP` neovlivňuje jakkoliv činnost serveru, pouze se klientovi odešle OK odpověď. Tento příkaz se využívá především u serverů, které implementují odpojování klientů při nečinnosti. Klient tedy, aby zabránil nechtěnému odpojení, periodicky odesílá serveru po určitém časovém intervalu příkaz `NOOP` a udržuje tak spojení aktivní.

2.4 Odpovědi FTP serveru

Klient po každém odeslaném příkazu serveru požaduje odpověď, určující stav operace či serveru. Pro každý příkaz musí server generovat minimálně jednu odpověď. Navíc u skupiny následujících příkazů

1 Více v kapitole 2.2

(jako např. RNFR a RNTO) odpovědi stanovují přechodný stav, pokud všechny předcházející příkazy proběhly v pořádku. Chyba v jakékoliv části vyžaduje odeslání celé sekvence příkazů znovu.

FTP odpověď sestává z tří číslic dlouhého kódu následovaného nějakým textem. Číslo slouží jako kód pro stavový automat klienta, kdežto text je určen jako informace pro člověka. Textová zpráva se tedy může server od serveru lišit.

Formát odpovědi serveru:

```
150 Some text for human user<CRLF>
```

Význam jednotlivých kódů je zobrazen v tabulce.

Kód	**Význam odpovědi.**
1xx	Předběžná pozitivní odpověď. Požadovaná akce byla zahájena, bude následovat ještě jedna odpověď stanovující ukončení akce.
2xx	Akce byla úspěšně dokončena.
3xx	Přechodná pozitivní odpověď. Použití u sekvencí příkazů (RNFR a RNTO), informuje o úspěšném provedení dosud obdržených příkazů.
4xx	Přechodná negativní odpověď. Příkaz nebyl úspěšný, ale klient jej může použít znovu, jelikož chyba je pouze dočasná.
5xx	Trvalá negativní odpověď. Příkaz nebyl úspěšný a klient by jej už neměl zkoušet používat.
x0x	Syntaxe – došlo k chybě v syntaxi.
x1x	Informace – odpověď na příkaz vyžadující nějakou informaci jako např. HELP.
x2x	Připojení – odpovědi související s kontrolním a datovým spojením.
x3x	Autentizace – odpovědi pro přihlašování.
x4x	Nespecifikováno.
x5x	Souborový systém – tyto odpovědi indikují stav serverového souborového systému.

Poslední číslice určuje jemnější odstupňování v jednotlivých funkčních skupinách specifikovaných druhou číslicí.

2.5 Příklad komunikace serveru a klienta

Následující tabulka zobrazuje průběh komunikace mezi klientem a serverem. Klient se připojí k serveru, přihlásí se uživatelským jménem tester a heslem heslo, vypíše si obsah aktuálního adresáře, stáhne si soubor foto.jpg a odpojí se. Datové spojení je navázáno jako aktivní tzn. server se připojí na adresu a port zaslaný klientem.

Příkaz	**Odpověď**
	220 Welcome
USER tester	

	331 User name okay, need password.
PASS heslo	
	230 User logged in, proceed.
PWD	
	257 "/"
TYPE A	
	200 Type is now ASCII.
PORT 192,168,2,104,10,72	
	200 Command okay.
LIST	
	150 File status okay; about to open data connection.
	226 Closing data connection.
TYPE I	
	200 Type is now BINARY.
PORT 192,168,2,104,10,80	
	200 Command okay.
RETR foto.jpg	
	150 File status okay; about to open data connection.
	226 Closing data connection.
QUIT	
	221 Goodbye

2.6 Zabezpečené verze protokolu FTP

Jelikož protokol FTP nedefinuje žádné zabezpečení přenosu dat po síti, je možné data odchytit a zneužít (např. jméno a heslo). Aby se tomuto zabránilo vzniklo postupně mnoho rozšíření zajišťujících zabezpečenou komunikaci. Zmíním pouze dvě nejvíce rozšířené, a to FTPS (někdy také FTP-SSL) a Secure FTP (nebo také FTP over SSH).

FTPS používá k šifrování spojení protokol transportní vrstvy TLS. Existují dva způsoby navázání šifrované komunikace. Prvním způsobem je způsob explicitní kdy si klient musí zabezpečený přenos vyžádat po připojení. Klient může určit zda se bude šifrovat pouze kontrolní spojení, pouze datové spojení nebo obě. Druhým způsobem je způsob implicitní kdy se klient nepřipojuje na port 21, ale na zabezpečený port 990. Při tomto způsobu je šifrována veškerá komunikace a klient nemá možnost toto změnit.

Secure FTP využívá pro zabezpečení přenosu tunelování pomocí protokolu SSH. Vzhledem k tomu, že FTP protokol používá více spojení, mohl by klient se serverem navázat datové spojení mimo tunel. Je tedy nutné, aby SSH klient měl znalosti o protokolu FTP, kontroloval zprávy posílané po kontrolním spojení a zajistil otevření datového spojení v rámci tunelu.

3 Platforma Windows Mobile

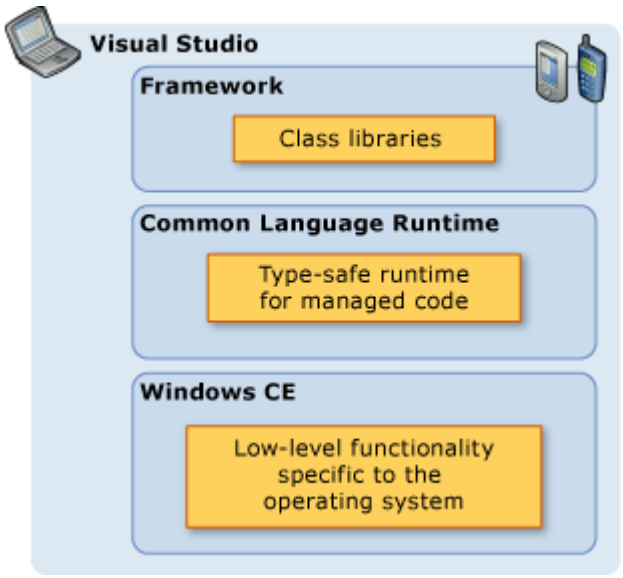
Windows Mobile je kompaktní operační systém od společnosti Microsoft, určený pro mobilní zařízení. Systém je navržen tak, aby alespoň částečně připomínal verzi operačního systému Windows pro stolní počítače. Aktuální verze systému v době psaní této práce je 6.1. V současné době jsou v distribuci tři edice systému a to Standard, Classic a Professional. Edice Standard je určena pro chytré mobilní telefony (smartphone) bez dotykového displaye, edice Classic je určena pro kapesní počítače (Pocket PC) s dotykovým displayem a edice Professional je určena pro kapesní počítače s dotykovým displayem a GSM modulem. Informace o platformě Windows Mobile jsou čerpány z [2].

Pro vývoj aplikací pro platformu Windows Mobile a prostředí .NET Compact Framework je nejlepší využívat přímo nástroje společnosti Microsoft, které jsou pro vývoj nejlépe vybaveny. Jedná se zejména o vývojové prostředí Microsoft Visual Studio. To nabízí veškeré prostředky pro vývoj jak malých tak i rozsáhlých projektů. Nabízí také kvalitní ladící nástroje určené k hledání chyb, IntelliSense sloužící k automatickému doplňování příkazů a spoustu dalších. Pro vývoj mobilních aplikací je třeba doinstalovat balík Windows Mobile SDK, ten kromě možnosti vývoje mobilních aplikací přidá například i emulátor mobilních zařízení v němž lze vyvíjené aplikace testovat a ladit.

3.1 Prostředí .NET Compact Framework

Microsoft .NET Compact Framework (dále .NET CF) je verzí .NET Frameworku pro mobilní zařízení. Obsahuje množství knihoven shodných s .NET Frameworkem a některé knihovny vlastní (určené například k specifickým operacím s mobilními a embedded zařízeními). Přesto jsou knihovny v .NET CF navrženy tak, aby zabíraly co nejméně místa. Aktuální verze .NET CF v době psaní této práce je 3.5, ovšem prostředí si udržuje zpětnou kompatibilitu, takže pokud pokud máte v zařízení nainstalován .NET CF 3.5, můžete pod ním provozovat i aplikace naprogramované pro verzi 2.0 nebo 1.0. Informace o prostředí .NET Compact Framework jsou čerpány z [3].

Prostředí je nezávislé na použitém hardware, takže je možné aplikace provozovat na různých platformách. Stejně tak je možné pro vývoj použít několik různých programovacích jazyků (např. C# nebo Visual Basic .NET) a to i v rámci jedné aplikace. Aplikace naprogramované v .NET se při překladu přeloží do CIL (Common Intermediate Language) byte kódu a až při spuštění se díky CLR (Common Language Runtime) a jeho Just-in-time kompilátoru přeloží do nativního kódu. Toto chování je možné pozměnit a vynutit přímý překlad do nativního kódu v nastavení projektu. Na obrázku 3.1 1 je vidět jak je prostředí .NET Compact Framework vystavěno.



Obrázek 3.1 1: Schéma architektury prostředí .NET Compact Framework

3.2 Jazyk C#

Jazyk C# je objektově orientovaný programovací jazyk vyvinutý společností Microsoft zároveň s platformou .NET. Projekt byl veřejně oznámen v červenci roku 2000 a později byl schválen jako standard sdruženími ECMA a ISO/IEC. Kompilátor C# od Microsoftu se drží implementace obou těchto standardů. Informace o jazyku C# jsou převzaté z [6], informace o jeho verzích jsou čerpány z [7].

C# je jazykem objektově orientovaným přesto však poskytuje podporu i pro komponentově orientované programování. Současný návrh software stále více závisí na komponentách ve formě sebe-popisujících balíčku funkcionality. Klíčem k takovýmto komponentám je přítomnost programového modelu s vlastnostmi, metodami, událostmi a připojenou dokumentací. C# poskytuje jazykové konstrukce, které přímo podporují tento koncept, což z něj dělá přirozený jazyk pro vývoj a použití softwarových komponent.

Jazyk C# má obecně mnoho rysů, které pomáhají při vývoji robustních aplikací. Patří mezi ně: Garbage collector, který se stará o uvolňování paměti zabírané nepoužívanými objekty. Správa výjimek, pro strukturovaný a rozšiřitelný přístup k detekci a zotavení z chyb. A typově bezpečný návrh jazyka, který například zabraňuje čtení neinicializovaných proměnných nebo přistupování do paměti mimo hranice pole. Dále má jazyk unifikovaný systém typů. Veškeré typy, včetně primitivních jako `int` a `double`, dědí z jediného kořenového objektu `object`. Čili všechny typy

sdílejí sadu společných operací a hodnoty jakéhokoli typu mohou být ukládány, přesouvány a prováděny nad nimi konzistentní operace. C# podporuje jak uživatelsky definované referenční typy, tak i hodnotové typy.

Aby se zajistilo, že se programy a knihovny napsané v C# mohou časem vyvíjet kompatibilním způsobem, byl kladen důraz na vložení verzování do návrhu jazyka. Mnoho programovacích jazyků tomuto problému věnuje malou pozornost a následkem toho se častěji než je nutné stává, že při uvolnění nových knihoven programy přestávají správně fungovat. Proto C# definuje modifikátory `virtual` a `override`, pravidla pro přetěžování metod a možnosti implementace rozhraní, které pomáhají tento problém řešit.

První verze programovacího jazyka C#, verze 1.0 byla vydána v roce 2002 zároveň s .NET Framework 1.0. Verze C# 2.0 byla vydána koncem roku 2005 a přidala například částečné a statické třídy, iterátory, anonymní metody a nullovatelné hodnotové typy. Zatím poslední verze C# 3.0 byla vydána na konci roku 2007 spolu s .NET Framework 3.5. Ta přidala například lambda výrazy, inicializátory objektů a kolekcí, rozšiřující metody, anonymní třídy, klíčové slovo var a výrazové stromy. Další plánovaná verze C# 4.0 je ve vývoji od října 2008 a datum jejího uvedení prozatím není známo. Měla by přinést například dynamické vyhledávání členů a volitelné parametry.

3.3 Síťová komunikace v C#

Jazyk C# poskytuje třídy pro práci s některými protokoly aplikační i transportní vrstvy referenčního modelu ISO/OSI. Většina těchto tříd je umístěna ve dvou jmenných prostorech, `System.Net` a `System.Net.Sockets`. `System.Net` poskytuje převážně třídy pro práci s protokoly aplikační vrstvy jako například HTTP či FTP, práci s webem či cookies, získávání informací z DNS apod. Zatímco `System.Net.Sockets` poskytuje třídy pro práci s protokoly transportní vrstvy. Informace o možnostech síťové komunikace v jazyce C# jsem čerpal z [13].

Vzhledem k zaměření této práce nás tedy bude více zajímat jmenný prostor `System.Net.Sockets`. Ten obsahuje třídu `Socket`, která implementuje Berkeley sockets (dále jako BSD). Tato třída nabízí prostředky pro implementaci jak spojované komunikace (TCP) tak i komunikace nespojované (UDP). Obsahuje například BSD metody `Bind`, `Listen`, `Accept`, `Connect`, `Send` a `Recieve`. Implementace serveru se provádí postupným provedením několika operací. Nejdříve se pomocí `Bind` sváže socket s lokálním koncovým bodem (síťové rozhraní, port), poté se pomocí `Listen` spustí naslouchání na daném socketu. Následně přijmeme připojeného klienta pomocí metody `Accept` a poté je již možné s ním komunikovat pomocí metod `Send` a `Recieve`. Implementace klienta je podstatně jednodušší. Je třeba pouze se pomocí metody `Connect` připojit k vzdálenému bodu a poté již komunikovat pomocí metod `Send` a `Recieve`.

Další možností implementace síťové komunikace je použít jiné třídy než `Socket` ze stejného jmenného prostoru, které však nabízí větší programátorskou přívětivost. Jedná se o třídy `TcpListener`, `TcpClient` a `UdpClient`. Jak již název napovídá tak třída `TcpListener` slouží k implementaci serveru. Její použití je velice jednoduché, stačí pouze v konstruktoru uvést lokální koncový bod (IP adresu rozhraní a port), spustit naslouchání pomocí metody `Start` a poté již přijmout pomocí metody `AcceptTcpClient` klienta definovaného instancí třídy `TcpClient`. `TcpClient` slouží k implementaci klienta a komunikaci přes síťové rozhraní. Pokud se pomocí třídy připojujeme k serveru stačí pouze použít metodu `Connect` jejíž parametr udává vzdálený koncový bod. Po připojení, ať už přes `TcpListener` nebo přímo pomocí `Connect`, můžeme přistupovat pomocí metody `GetStream` přímo k síťovému proudu (`NetworkStream`) z nějž lze již jednoduše číst a do něj zapisovat. Tento proud na rozdíl od jiných proudů v C# není bufferovaný, tedy veškerá data zapsaná do tohoto proudu jsou ihned odeslána.

Je tedy zřejmé, že v jazyce C# existuje několik způsobů implementace síťové komunikace a je pouze na programátorovi, který přístup využije.

3.4 Vícevláknové aplikace v C#

V dnešní době je potřeba programovat vícevláknové aplikace dost vysoká a v mnoha ohledech se na ni klade velký důraz. Díky příchodu více-jádrových procesorů se tato potřeba ještě více prohloubila, hlavně kvůli snaze využívat jejich plný výkon. Programování vícevláknových aplikací přináší spoustu výhod, ale také nevýhod. Mezi zřejmé výhody patří zpracovávání několika operací současně, například současné získávání vstupu od uživatele, překreslování výstupu v okně a získávání vstupu ze sítě. Nevýhodou je zejména problém synchronizace vláken s kterým přímo souvisí problém uváznutí. Informace o prostředcích jazyka C# pro tvorbu vícevláknových aplikací jsou čerpány z [13].

V jazyce C# jsou třídy pro práci s vlákny a jejich synchronizaci umístěny ve jmenném prostoru `System.Threading`. Jedná se kromě jiných o třídu `Thread`, která slouží k vytvoření a ovládání nového vlákna a třídy `Mutex`, `Monitor` a `Semaphore`, sloužící k synchronizaci vláken.

Vytvoření nového vlákna za pomoci třídy `Thread` se provádí předáním delegáta (ukazatel na funkci) typu `ThreadStart` konstruktoru třídy. Poté již stačí zavolat metodu `Start` a nové vlákno nám již běží. Toto platí pro neparametrizovaná vlákna, pokud potřebujeme funkci pracující v jiném vlákně při jejím startu předat nějaké parametry je třeba použít delegát typu `ParameterizedThreadStart`, funkci definovat jeden parametr typu `object` a hodnotu tohoto objektu předat jako parametr metody `Start`. Po spuštění nového vlákna máme například možnost

vlákno násilně ukončit pomocí metody `Abort`, uspat pomocí metody `Sleep`, počkat na jeho ukončení pomocí metody `Join` a spoustu dalších.

Ovšem po vytvoření nového vlákna v mnoha případech práce nekončí, je třeba zabezpečit sdílená data a k tomu slouží výše uvedené třídy `Mutex`, `Monitor` a `Semaphore`. Třída `Mutex` poskytuje metody sloužící k uzamčení sekce kódu, ke které může přistupovat v daný okamžik pouze jedno vlákno. Nejdříve se zavolá metoda (např. `WaitOne`) pro uzamčení bloku kódu a poté je třeba aby stejné vlákno, které blok uzamklo jej zase odemklo (kontroluje se identifikace vlákna) pomocí `ReleaseMutex`. Je možné vytvořit i pojmenovaný `Mutex`, který je poté dostupný také ostatním aplikacím, lze tedy pomocí něj synchronizovat i dva a více procesů. Další možností jak omezit přístup do určité části kódu pouze jednomu vláknu je použít třídu `Monitor`. Pro uzamčení přístupu slouží metoda `Enter` a pro odemčení metoda `Exit`. Existuje však i další, jednodušší, možnost než použití `Enter` a `Exit`, a to klíčové slovo `lock`, které povoluje uzamknout blok kódu právě pomocí monitoru. Pokud má mít do kritické sekce povolen přístup více vláken lze použít třídu `Semaphore` a v jejím konstruktoru určit maximální počet vláken v kritické sekci. Poté pomocí metody `WaitOne` vstoupit do kritické sekce a metodou `Release` ji opustit.

Vlákna mohou mnohé činnosti zrychlit a zjednodušit, ale velice důležitou roli hraje zajištění správné synchronizace vláken a to leží na bedrech programátora.

3.5 Práce s XML v C#

XML (z anglického eXtensible Markup Language) je obecný značkovací jazyk, který umožňuje vytváření konkrétních značkovacích jazyků pro rozsáhlé spektrum použití. XML se dá využívat od ukládání nastavení až po ukládání dokumentů, například nový formát Office Open XML společnosti Microsoft použitý pro ukládání dokumentů v Microsoft Office 2007 je založený mimo jiné na jazyce XML. Samotné vývojové prostředí Microsoft Visual Studio tento značkovací jazyk hojně využívá, konfigurační soubory projektů, integrovaný dokumentační systém, uživatelské nastavení, to vše je uloženo (zapsáno) v XML. Informace o tvorbě a přístupu k XML dokumentu v jazyce C# jsou čerpány z [13].

V jazyce C# existuje mnoho způsobů vytváření a čtení XML dokumentů. Od těch nejprimitivnějších, jako je přistupování k XML jako textovému dokumentu a jeho následné složité rozebírání, přes práci s XML dokumentem pomocí DOM (Document Object Model), až po vysoce úrovňové jako je serializace. Zatímco první způsob svědčí hlavně o špatném přístupu programátora, další dva nabízí kvalitní nástroje pro práci s XML dokumentem.

Pro práci s XML dokumentem pomocí DOM slouží ve jmenném prostoru `System.Xml` umístěné třídy `XmlDocument`, `XmlNode`, `XmlElement`, `XmlAttribute`, `XmlDeclaration` a spousty dalších. `XmlDocument` implementuje mimo jiné metody `Load`, `Save`, `SelectSingleNode`, `SelectNodes`, `CreateElement` a `AppendChild`. Pomocí metody `Load` je možné načíst XML dokument ze souboru, nebo i z URL, kdežto metoda `Save` slouží k uložení stromu XML dokumentu do souboru. Po načtení dokumentu můžeme pomocí metody `SelectSingleNode` a jazyku XPath[2] přistupovat k jednotlivým uzlům či elementům dokumentu. Nebo pomocí metody `SelectNodes` a jazyku XPath přistupovat ke skupině uzlů či elementů. Pro přidávání nových uzlů či elementů slouží metoda `CreateElement`, pomocí ní lze vytvořit nový objekt typu `XmlElement` a ten poté pomocí metody `AppendChild` připojit do kořene stromu dokumentu, nebo k jinému uzlu. Jedná se tedy u větších a složitějších dokumentů o vcelku složitý přístup.

Další možností je využít takzvanou serializaci. Jedná se v podstatě o uložení objektu na paměťové médium v zadaném formátu (v našem případě XML). K serializaci do a z XML slouží třída `XmlSerializer` umístěná ve jmenném prostoru `System.Xml.Serialization`. Serializace objektu do XML souboru se provádí pomocí metody `Serialize`, které předáme parametrem příslušný `TextWriter` (jedná se v podstatě o objekt reprezentující soubor otevřený pro zápis) a objekt který serializujeme. O načtení objektu z XML se stará metoda `Deserialize`, té předáme příslušný `TextReader` (objekt reprezentující soubor otevřený pro čtení) a následně nám vrátí získaný objekt, tento je již pouze potřeba přetypovat na námi požadovaný typ a poté je možné s ním pracovat. Při serializaci se názvy jednotlivých uzlů automaticky stanoví podle názvů tříd a vlastností (properties) objektu, toto chování je možné změnit pomocí zapsání formátovacího řetězce před definici třídy, vlastnosti, atd. Stejně tak je možné i stanovit zda bude vlastnost zapsána jako element, nebo jako atribut apod. Programátor nemusí stanovovat strukturu XML dokumentu. Ta je, pokud není dáno jinak, stanovena automaticky podle struktury serializovaného objektu. Serializace tedy nabízí jednoduchý a dobře modifikovatelný přístup k XML.

Jak je vidět, tak jazyk C# nabízí několik různých způsobů jak pracovat s XML a záleží jen na programátorovi, který způsob si vybere a je pro jeho aplikaci výhodnější.

2 Jazyk sloužící k adresaci částí XML dokumentu

4 Popis návrhu a implementace

Tato kapitola se zabývá popisem návrhu a implementace samotné aplikace FTP serveru pro Windows Mobile verze 6. Postupně se pokusím detailně popsat a vysvětlit jednotlivé části aplikace.

Aplikace byla při vývoji rozdělena do několika různých projektů. Prvním z nich je projekt *Shared*, ten obsahuje sdílené a pomocné třídy (pomocná třída pro práci se sockety, třída s nastavením, atd.), jeho výstupem je poté dynamicky linkovaná knihovna *Shared.dll*. Dalším je projekt s názvem *FtpHandlers*, ten obsahuje veškerou logiku FTP serveru (obsluha řídícího a datového spojení, zpracování příkazů), jeho výstupem je dynamicky linkovaná knihovna *FtpHandlers.dll*. Dalším projektem je *Application*, ten reprezentuje prezentační vrstvu aplikace (dialogy, komunikace s uživatelem, apod.), jeho výstupem je spustitelný soubor *WMFtpServer.exe*. Dalším projektem je *SmartphoneDialogs*, dynamicky linkovaná knihovna implementující dialog pro procházení a otevírání složek, který není v prostředí .NET Compact Framework implementován. Implementace této knihovny byla převzata z ukázkové aplikace [8], licenční smlouva je přiložena ve složce projektu. Posledním projektem je *InstallCab*, tento projekt vytvoří z výsledné aplikace instalační balíček pro mobilní zařízení. Díky takto rozvržené aplikaci je možné nad knihovnami *Shared.dll* a *FtpHandlers.dll* vystavět jiné uživatelské rozhraní FTP serveru pro mobilní zařízení bez dostupnosti zdrojového kódu.

V následujících podkapitolách postupně nastíním průběh návrhu, vysvětlím implementaci jednotlivých logických celků aplikace a možnost použití knihoven *Shared.dll* a *FtpHandlers.dll* v jiné aplikaci.

4.1 Síťová část

Tato podkapitola se věnuje síťové části aplikace, tedy zejména přijímání klientů, správě řídícího spojení a správě datového spojení.

Samotné implementaci předcházelo důkladné prostudování dokumentace k protokolu FTP [1] a snaha o pochopení jeho síťového rozhraní. Následně bylo možné se zabývat rozvržením jednotlivých částí a pustit se do implementace.

Síťová část je rozdělena do několika tříd ve jmenném prostoru `Ftp.FtpHandlers`. Jedná se především o třídy `ServerHandler`, `ClientHandler` a `DataConnection`.

Třída `ServerHandler` implementuje logiku serveru - naslouchá na zadaném portu a přijímá nové klienty. Svou činnost provádí v samostatném vlákně, hlavním důvodem pro implementaci naslouchání v samostatném vlákně bylo využití blokujících socketů. Bylo tedy třeba zabezpečit aby

spuštění serveru neblokovalo samotnou aplikaci. Při návrhu byl velký důraz kladen na zapouzdření objektu. Výsledkem je, že programátorovi jsou přístupné pouze dvě metody a jedna vlastnost, pomocí nichž může ovládat celý server. Jedná se o metody `Start`, `Stop` a vlastnost `ClientsCount`. Metoda `Start`, jak již její název napovídá, slouží ke spuštění FTP serveru, má jeden nepovinný parametr a to číslo portu na kterém má server naslouchat. Přesněji řečeno metoda spustí v novém vlákně naslouchání, kdy je po připojení nového klienta inicializován a spuštěn objekt třídy `ClientHandler` reprezentující tohoto klienta, ten je poté uložen do seznamu a pokračuje se dále v naslouchání. Také se provádí kontrola maximálního počtu připojených klientů, jakmile je tohoto počtu dosaženo, tak server automaticky odešle klientovi zprávu o zaplnění serveru a klienta odpojí ještě před inicializací jeho objektu. Metoda `Stop` slouží k ukončení naslouchání a ukončení vlákna serveru, také jsou po jejím zavolání násilně odpojeni všichni aktuálně připojení klienti. Vlastnost `ClientsCount` slouží jako informativní položka, která vrací počet aktuálně připojených klientů.

Třída `ClientHandler` reprezentuje připojeného klienta a v podstatě se jedná o správu jeho kontrolního spojení. Vzhledem k požadavku na konkurenčnost serveru je i zpracování klienta prováděno v samostatném vlákně. I při návrhu klienta se stejně jako při návrhu serveru dbalo na zapouzdření objektu. Díky tomu třída implementuje pouze dvě veřejné metody, těmi jsou `Start` a `Stop`. Metoda `Start` slouží ke spuštění nového vlákna a načítání příkazů z řídícího spojení, kdežto metoda `Stop` slouží k ukončení práce s klientem, jeho odpojení a ukončení příslušného vlákna. Při ukončování metoda vyvolá událost, která signalizuje serveru, že byl klient ukončen a server si jej může odstranit ze svého seznamu. V pracovním vlákně probíhá mimo načítání příkazů také jejich analýza a volání příslušných metod příslušného objektu pro vykonání daného příkazu, ale o tom více v kapitole 4.2. Třída také implementuje větší množství různých vlastností, ty slouží především pro udržování stavu řídícího spojení a vykonávání příkazů od klienta. Jednou z těchto vlastností je i instance třídy `DataConnection`.

Třída `DataConnection` implementuje metody pro navázání, správu a uzavření datového spojení. Jedná se o metody `StartListening`, `AcceptClient`, `StopListening`, `Connect` a `Close`. V případě, že je požadováno pasivní spojení, je nejdříve zahájeno naslouchání pomocí metody `StartListening` a následně je pomocí metody `AcceptClient` přijat klient, s kterým je již možné komunikovat. Pokud je požadováno spojení aktivní použije se metoda `Connect` pro připojení ke vzdálenému koncovému bodu. Metody `StopListening` a `Close` slouží k ukončení naslouchání a uzavření spojení mezi serverem a klientem. Důležitá je hlavně vlastnost `Socket`, která po navázání spojení pomocí výše uvedených metod poskytuje přístup k objektu typu `TcpClient` reprezentujícímu datové spojení.

Do síťové části by se dala zařadit i třída `SocketFunctions` ze jmenného prostoru `Ftp.Shared`, která nabízí množství statických metod pro čtení a zápis dat na zadaný socket typu `TcpClient`, nebo také například statickou metodu pro zjištění vlastní IP adresy.

Přesnější informace o implementaci a použití jednotlivých tříd a metod se dají zjistit nahlédnutím do zdrojových kódů a jejich komentářů.

4.2 Zpracování příkazů protokolu FTP

Návrh a implementace zpracování příkazů protokolu FTP byla jedna z časově nejnáročnějších částí. Samotnému návrhu předcházelo důkladné prostudování chování a použití jednotlivých příkazů a vzhledem k velmi volné interpretaci dokumentace k protokolu FTP [1] u některých příkazů, bylo také nutné prostudovat chování alespoň malého vzorku dnešních FTP klientů. Po nastudování všech těchto informací bylo možné přejít k návrhu. Nejdříve bylo nutné vyřešit reprezentaci jednotlivých příkazů, vzhledem k objektovosti jazyka C# bylo řešení nasnadě. Každý příkaz je tedy reprezentován třídou děděnou od společné třídy `CommandHandler`. Ta definuje základní metody a vlastnosti pro každý implementovaný příkaz. Jednotlivé příkazy i s třídou `CommandHandler` jsou umístěny ve jmenném prostoru `Ftp.FtpHandlers.Commands`, celkem je implementováno 25 různých příkazů protokolu.

Otcovská třída pro všechny příkazy, `CommandHandler`, definuje dvě základní metody a několik vlastností. Metodami jsou `ExecuteCommand` a `Execute`, první z nich se volá při požadavku na provedení příkazu, jejím parametrem jsou parametry příkazu. Tato metoda provádí kontrolu oprávnění k uskutečnění operace (zda je uživatel přihlášen) a podle jejího výsledku volá metodu `Execute` nebo skončí. Druhá metoda je definovaná jako virtuální a slouží k vykonání samotného příkazu. To znamená, že jednotlivé třídy reprezentující příkazy, které od této třídy dědí, musí tuto metodu předefinovat požadovaným chováním. Při pokusu o volání nepředefinované metody `Execute` je vyvolána výjimka a je ukončeno vykonávání aktuálního příkazu. Dále třída definuje několik vlastností, například velikost bufferu pro datové spojení, nebo také vlastnost udržující referenci na klientský objekt, pomocí níž je možné měnit stav řídícího spojení.

Další otázkou, kterou bylo nutno řešit, byl způsob volání jednotlivých příkazů. Jako řešení se nabízela rozsáhlá konstrukce switch, ovšem tato se nejevila příliš esteticky pěkně a ani nebyla příliš přístupná budoucím úpravám (např. přidávání dalších příkazu). Nakonec jsem tedy zvolil poněkud jiný přístup. Tím je inicializace objektů jednotlivých příkazů a jejich uložení do hashovací tabulky, kde klíčem je název příkazu a hodnotou reference na objekt. Provedení příkazů se tedy vykoná vyhledáním daného příkazu v hashovací tabulce a zavoláním příslušné metody získaného objektu.

Tato konstrukce usnadnila jak vyhledávání příkazů, tak i jejich přidávání, ale hlavně zpřehlednila kód.

Implementace většiny příkazů nebyla i přes časovou náročnost příliš složitá. Pouze některé příkazy přinesly dodatečné potíže. Například příkaz `LIST` jsem musel přepracovávat hned několikrát, důvodem byla skutečnost, že dokumentace protokolu neuvádí formát přenášených dat. Postupně se mi podařilo na internetu najít několik údajně podporovaných formátů, ale bohužel až po jejich implementaci jsem zjistil, že těmto formátům dnešní klienti nerozumí. Nakonec se mi naštěstí podařilo najít formát */bin/ls*[3], který je dnešními klienty podporován.

Jednotlivé příkazy jsou implementovány ve třídách začínajících slovem `Command` následovaným jménem příkazu, například `CommandList`, `CommandHelp`, `CommandStor` atd. Každá z nich předefinovává funkci `Execute` otcovské třídy, tak aby reprezentovala chování daného příkazu.

4.3 Práce se souborovým systémem

Práce se souborovým systémem u mobilních zařízení je oproti systémům stolním poněkud odlišná. Hlavní rozdíl je v používaném formátu cest. Zatímco na stolních systémech Windows je kořenovým elementem cesty písmeno jednotky, u mobilních systému Windows Mobile je kořenovým elementem cesty stejně jako v Unixových operačních systémech znak `\`. Jednou z dalších věcí, kterou bylo při návrhu aplikace zohlednit, je že u jakéhokoli příkazu protokolu FTP, který jako parametr přijímá název souboru, je možné zadat jak relativní tak i absolutní cestu. Navíc různí klienti používají pro pohyb mezi adresáři různé příkazy a cesty, jedná se hlavně o způsob vynořování (přesun v adresářové struktuře nahoru), kdy někteří klienti zadávají absolutní cestu, někteří jako cestu zadají `..` a setkal jsem se dokonce i s klienty, kteří používají kombinaci obou způsobů.

Jelikož je možné nastavit každému uživateli jeho výchozí (bázový) adresář je tedy nutné rozlišovat cesty na úrovni FTP a systému. Cesta na úrovni FTP udává relativní cestu z bázového adresáře na úrovni systému. Jazyk C# nabízí přímo metodu na spojování a kombinování dvou cest. Jedná se o metodu `Path.Combine`, ta vrací kombinaci dvou cest zadaných parametry. Ovšem vzhledem k podobnosti cest pro FTP a mobilní zařízení (jediný rozdíl je v použití opačných lomítek), nastával problém při zadání druhé cesty jako absolutní. Pokud byla FTP cesta zadána jako absolutní, metoda `Path.Combine` vrátila místo očekávané kombinace pouze cestu FTP. Proto bylo nutné nejdříve cestu převést a upravit do správného formátu.

Veškerá funkcionalita pro práci se souborovým systémem je implementována v třídě `FileManipulation` ve jmenném prostoru `Ftp.Shared`. Instance této třídy je vytvářena

3 Více o tomto formátu v kapitole 2.3.3

v klientském objektu ihned po připojení uživatele. Třída si udržuje jak bázovou cestu (systémovou) uživatele, která zůstává stejná po celou dobu připojení uživatele, tak i aktuální adresář (FTP), který se mění v závislosti na požadavku uživatele. Třída poskytuje dvě veřejné vlastnosti, těmi jsou aktuální cesta v FTP serveru a aktuální cesta v souborovém systému zařízení. Dále nabízí několik statických metod sloužících k zjištění zda je na zadané cestě umístěn soubor či adresář, nebo zda zadaná cesta existuje. Třída implementuje také metody pro vytvoření adresáře, smazání adresáře, smazání souboru, přejmenování zadaného souboru či adresáře apod. Nebo i metodu, která vrací seznam položek (včetně jejich vlastností) zadaného adresáře. Metody pro pohyb adresářovým systémem, které mění aktuální adresář. Ale hlavně implementuje metodu `GetPath`, která za pomocí úpravy zadaných cest a metody `Path.Combine` získá absolutní systémovou cestu pro zadanou cestu relativní.

Většina implementované funkcionality by měla být schopná za normálních okolností fungovat i na stolním systému. Výjimkou je snad pouze statická metoda pro získání adresáře v němž je umístěna aplikace, kde je použit specifický způsob získání cesty pro mobilní zařízení.

4.4 Ukládání nastavení

Jelikož si aplikace potřebuje udržovat svoje nastavení nejenom během svého chodu, ale i po ukončení, je třeba na toto při návrhu myslet. Navíc k tomuto nastavení potřebuje přístup jak vlákno každého klienta, tak i vlákno serveru a uživatelské rozhraní aplikace. Je tedy třeba zajistit, aby potřebné informace byly dostupné všem. O toto se stará třída `SettingsData` ze jmenného prostoru `Ftp.Shared`. Třída je implementována pomocí návrhového vzoru *Singleton* (více o návrhových vzorech např v [14]), díky čemuž je zajištěno, že v celé aplikaci bude dostupná pouze jedna její instance a nebude docházek k zbytečné redundanci dat a problémy se synchronizací. K instanci třídy se přistupuje pomocí statické metody `Instance`, následně je již možné zjišťovat popřípadě měnit jakékoliv nastavení. Třída udržuje informace o maximálním počtu souběžně připojených klientů, číslo portu na kterém server naslouchá, informaci o povolení či zakázání anonymního přístupu k FTP serveru a seznam uživatelů. Seznam uživatelů je implementován pomocí hashovací tabulky, kde klíčem je jméno uživatele a hodnotou struktura obsahující jeho heslo a bázovou cestu. Třída definuje několik metod pro práci se seznamem uživatelů jako například přidání uživatele, úprava uživatele, kontrola hesla, zjištění existence uživatele a mnoho dalších metod, které usnadňují práci s nastavením. Toto se ovšem týká pouze udržování nastavení za chodu aplikace, aby bylo možné udržovat nastavení i po jejím vypnutí, bylo nutné implementovat i metody pro uložení a načtení z paměťového média. Důležité bylo stanovit v jakém formátu se bude nastavení ukládat. Řešení se nabízelo hned několik, mezi nimi například databáze a XML. Nakonec jsem díky jeho dobré čitelnosti a přenositelnosti zvolil formát XML.

K uložení a načtení nastavení slouží metody `Save` a `Load`. Nakonec byl pro implementaci práce s XML zvolen přístup pomocí DOM (Document Object Model). Serializaci nebylo bohužel možné jednoduše využít díky nedostupnosti třídy `DictionaryBase`, pro definici kolekce uživatelů, v .NET Compact Framework. Formát XML souboru s nastavením je definován následujícím DTD (více o DTD v [12]):

```
<!DOCTYPE wmftp [
 <!ELEMENT wmftp (users? , settings?)>
 <!ELEMENT users (user* , anonymous?)>
 <!ELEMENT settings (port? , maximalusers?)>
 <!ELEMENT user EMPTY>
 <!ELEMENT anonymous EMPTY>
 <!ELEMENT port (#PCDATA)>
 <!ELEMENT maximalusers (#PCDATA)>
 <!ATTLIST user
       name ID #REQUIRED
       pass CDATA #REQUIRED
       path CDATA #REQUIRED>
 <!ATTLIST anonymous
       enabled (false|true) #REQUIRED
       path CDATA #REQUIRED>
]>
```

Výsledný XML dokument pak může vypadat například takto:

```
 <wmftp>
     <users>
         <user name="tester" pass="test" path="\Temp" />
         <anonymous enabled="true" path="\My Documents" />
     </users>
     <settings>
         <port>21</port>
         <maximalusers>5</maximalusers>
     </settings>
 </wmftp>
```

Význam jednotlivých elementů XML dokumentu je velice jednoduché odvodit od jejich názvu, takže není třeba je více rozebírat. Pouze bych uvedl, že jak je na příkladu vidět, tak heslo je ukládáno v prosté textové podobě. Je to tak proto, aby mohl správce serveru heslo jednoduše v případě potřeby zjistit (v dialogu se zobrazuje také jako prostý text). Navíc jelikož není implementována podpora TLS/SSL a zabezpečení řídícího a datového spojení, je možné heslo bez problému odposlechnout.

4.5 Uživatelské rozhraní

Při návrhu uživatelského rozhraní pro mobilní aplikace je třeba brát v úvahu zejména malou velikost displaye mobilního zařízení. Do okna se tedy nevejde příliš informací a je třeba je logicky rozdělit tak, aby ovládání aplikace bylo co neintuitivnější (i když toto neplatí pouze u mobilních aplikací). Také je dobré při návrhu počítat s možností přepínání zobrazení mezi takzvanými landscape a portrait módy u mobilního zařízení.

4.5.1 Hlavní dialog

Již v rané fázi návrhu uživatelského rozhraní jsem plánoval, že hlavní dialogové okno bude sloužit hlavně ke spuštění a zastavení FTP serveru a zobrazování souhrnných informací o jeho stavu a činnosti.

Hlavní dialogové okno obsahuje tlačítko sloužící k spuštění a zastavení serveru. Popisek tohoto tlačítka se mění v závislosti na tom zda je server spuštěn (*online*) či zastaven (*offline*). Dále je zde uvedeno několik pro uživatele užitečných informací. Jsou to počet aktuálně připojených klientů, doba po kterou je server spuštěn (ve formátu HH:mm), IP adresa pro připojení k serveru a číslo portu na němž server naslouchá. Tyto informace se periodicky obnovují každých deset vteřin. Přes položky menu je možné zobrazit dialogy pro nastavení aplikace a uživatelů.

4.5.2 Dialog nastavení

Dialog pro nastavení slouží k nastavení základních parametrů serveru. Jedná se o číslo portu, na kterém má server naslouchat, maximální počet souběžně připojených klientů a povolení či zakázání anonymního přístupu.

Pole pro zadání čísla portu a maximálního počtu klientů jsou vytvořena pomocí ovládacího prvku `NumericUpDown`. Vstupní hodnoty těchto prvků jsou omezeny na 1 až 65535 pro číslo portu a 1 až 100 pro maximální počet připojených klientů. Pro povolení či zakázání přístupu anonymních uživatelů slouží zaškrtávací políčko, které v závislosti na svém stavu povolí, či zakáže zadání adresáře s výchozí cestou pro anonymní přístup. Aby nebylo nutné zapisovat cestu ručně je možné

využít tlačítka *Procházet* pro otevření dialogu pro výběr složky. Zadaná cesta je před uložením hodnot zkontrolována a dialog nepovolí uložení neexistující cesty.

4.5.3 Dialog správy uživatelů

Dialog správy uživatelů slouží k výpisu seznamu uživatelů, jejich hesel a přiřazených cest. Původně mělo být v tomto dialogu přítomné i nastavení anonymního přístupu, které je umístěné v dialogu s nastavením. Avšak vzhledem k malé velikosti dialogu a kvůli tomu, že způsobovalo horší čitelnost prvku pro zobrazení seznamu uživatelů (musel být podstatně menší), jsem se rozhodl pro jeho přesun.

Pro zobrazení seznamu uživatelů je použit prvek `ListView`, ten například dovoluje uživateli měnit velikost jednotlivých sloupců a po implementační stránce je práce s ním také příjemná. Zobrazeny jsou celkem tři informace, jméno uživatele, heslo v textové podobě a nastavená cesta. Pomocí tlačítka *Přidat* je možné otevřít dialog pro přidání nového uživatele, výběrem záznamu (uživatele) v seznamu a stisknutím tlačítka *Upravit* je možné vyvolat dialog pro editaci daného záznamu. A nakonec výběrem záznamu v seznamu a stisknutím tlačítka *Odstranit* je možné daný záznam odstranit. V tomto případě se zobrazí okno pro potvrzení odstranění daného záznamu.

4.5.4 Dialogy pro přidání a úpravu uživatele

Dialogy pro přidání a úpravu uživatele jsou v podstatě stejné, jediným rozdílem je nemožnost úpravy uživatelova přihlašovacího jména v dialogu pro úpravu.

Dialogy obsahují položku pro zadání přihlašovacího jména, hesla v textové podobě a zadání příslušné výchozí cesty uživatele. Pro zadání cesty je možné využít dialogu pro výběr složky, který je dostupný pomocí tlačítka *Procházet*. Před uložením zadaných dat je nejdříve otestována jejich validita, tzn. zda uživatel již neexistuje (v případě přidání), zda zadaná cesta existuje či zda nebylo zadáno prázdné heslo. Po zjištění validity zadaných dat se data uloží, je provedena změna hodnoty v seznamu uživatelů v dialogu správy uživatelů, vypsána informace o úspěšně provedené operaci a dialog je ukončen.

4.5.5 Dialog pro výběr složky

Vzhledem k nedostupnosti dialogu pro výběr složky v prostředí .NET Compact Framework, jsem byl nucen využít cizí komponentu tento dialog implementující. Implementace byla převzata z ukázkové aplikace pro práci se souborovým systémem na mobilních zařízeních z [8]. Provedl jsem pouze změny umožňující lokalizaci dialogu a odstranění přebytečného kódu. Licenční smlouva pro použití ukázkové aplikace z níž je dialog převzat je umístěna ve složce projektu tohoto dialogu (*SmartphoneDialogs*).

4.6 Využití implementovaných knihoven

Jak již bylo výše uvedeno je za pomocí dynamicky linkovaných knihoven *Shared.dll* a *FtpHandlers.dll* možné jednoduše vytvořit aplikaci využívající implementaci FTP serveru právě z těchto knihoven. Knihovny by měly být použitelné nejenom v aplikaci mobilní, nýbrž i v aplikaci stolní, ovšem toto použití nebylo zcela otestováno, takže nelze zaručit plnou funkčnost.

Jejich použití je velice jednoduché, stačí pouze do svého projektu přidat reference na dané dvě knihovny a poté je již možné využívat jimi implementované třídy a jejich metody. Pro spuštění a zastavení serveru slouží metody třídy `ServerHandler` `Start` a `Stop` ze jmenného prostoru `Ftp.FtpHandlers`. Před samotným spuštěním serveru však doporučuji, inicializovat objekt třídy `SettingsData` (jmenný prostor `Ftp.Shared`) a nahrát do něj například nastavení z určitého souboru. Pokud nebude třída `SettingsData` inicializována, tak se nic neděje, třída serveru si ji zinicializuje sama s výchozími hodnotami (port 21, maximálně 10 aktivních uživatelů, anonymní přístup zakázán), ale nebude možné se díky výchozímu nastavení k serveru připojit. Poté je již server schopen operovat samostatně a není třeba dalších zásahů. Ovšem jsou dostupné i další možnosti například v kapitole 4.1 zmíněná vlastnost `ClientsCount`, nebo je možné si zaregistrovat událost `SendMessageEvent` ze třídy `GlobalMessaging` ve jmenném prostoru `Ftp.Shared`, která slouží k přenosu informací o chybách ke kterým došlo v rámci běhu serveru.

Veškeré veřejné metody jednotlivých tříd jsou patřičně okomentovány, a díky kompilaci komentářů do výsledného kódu je jejich použití a pochopení v mnoha případech opravdu jednoduché. Jak je vidět lze tedy za pomocí těchto knihoven a pár řádků kódu postavit aplikaci implementující logiku a funkčnost FTP serveru.

5 Testování

Testování je jednou z významných částí životního cyklu software. Ať už se jedná o program sčítající dvě čísla nebo rozsáhlou komerční aplikaci, je třeba výsledný produkt řádně otestovat, jinak se můžeme dočkat nepříjemných následků. I v tomto případě byla výsledná aplikace otestována s několika klienty a v pozdější fázi i na několika zařízeních.

Komunikace byla testována zejména s následujícími klienty:

- Total Commander 7.04a – dostupný na http://www.ghisler.com
- Total Commander for Pocket PC 2.51 – dostupný na http://www.ghisler.com
- Mozilla Firefox 3.0.9 – dostupný na http://firefox.czilla.cz/
- FireFtp 1.0.4 – dostupný na http://fireftp.mozdev.org/

Při testování jsem se hlavně zaměřil na správné fungování serveru a provádění jednotlivých příkazů. Testovalo se například procházení adresářové struktury, zda jsou soubory po přenosu identické s originálem nebo autentizace uživatelů. Testy zjišťující výkonnost aplikace prováděny nebyly, ale sledoval jsem alespoň maximální přenosové rychlosti. Ty se pohybovaly řádově v desítkách kB/s na emulátoru a v jednotkách MB/s v případě testování na stolním operačním systému.

Během tohoto testování se podařilo lokalizovat a napravit několik problémů s používáním různého zápisu cest různými klienty.

Server byl také testován na dvou verzích operačního systému Windows Mobile. Prvním byla verze 6.0 v emulátoru zařízení a druhým verze 6.1 na mobilním telefonu MIO A701.

V pozdější fázi vývoje se provádělo testování i na stolním operačním systému Windows XP. Tam bohužel nelze, z důvodu rozdílného použití funkce pro zjištění umístění aplikace, uložit nastavení. A z důvodu absence dialogu pro procházení složek v prostředí .NET Compact Framework a jeho nahrazení jinou komponentou navrženou speciálně pro mobilní zařízení není možné otevřít dialog *Procházet*.

6 Závěr

Výsledkem této práce je, dovolím si říci, konkurenceschopná aplikace FTP serveru pro platformu Windows Mobile. Server nabízí přehledné uživatelské rozhraní, ale hlavně veškeré funkce nutné pro bezproblémovou komunikaci s většinou dnešních klientů. Díky vytvořenému instalačnímu balíčku a snadnému nastavení je i jeho zprovoznění na mobilním zařízení záležitostí několika málo minut.

Během tvorby této aplikace jsem získal spoustu cenných zkušeností s vývojem aplikací pro mobilní zařízení a prostředím .NET Compact Framework. Zdokonalil jsem se v práci s jazykem C#, vývojem síťových a vícevláknových aplikací a procvičil jsem se v návrhu objektových aplikací.

Při vývoji a návrhu aplikace jsem využil znalostí z několika různých předmětů. Z předmětu ISA – Síťové aplikace a správa sítí, jsem díky projektu věděl jak pracuje FTP server a jakou cestou se při jeho implementaci vydat. Předmět IW5 – Programování v .NET a C# mi poskytl dostatečné teoretické i praktické základy pro vývoj aplikací v jazyce C#. Předmět ITU – Tvorba uživatelských rozhraní mi pomohl při návrhu vzhledu a ovládání aplikace.

Při testování aplikace jsem zjistil, že i když se něco může zdát hotové, zpravidla tomu tak není. Díky testování se mi podařilo odhalit několik více i méně závažných problému, ale hlavně jsem pochopil, že špatně otestovaný program je většinou špatný program.

Možností pro budoucí rozšíření aplikace je díky různým rozšířením protokolu FTP hned několik. Jedná se zejména o další příkazy a aliasy pro příkazy stávající. V budoucnu snad bude možné přidat podporu pro zabezpečený přenos, kterou nebylo možné implementovat díky nedostupnosti tříd a metod pro zabezpečenou komunikaci v aktuálních verzích prostředí .NET Compact Framework (verze 2.0 a 3.5). To se snad v budoucnosti změní. Další možností vylepšení je přidání automatického odpojování nečinných klientů po daném časovém intervalu pro uvolnění prostředků serveru, nebo detailnější možnosti nastavení přístupových práv jednotlivých klientů.

Literatura

[1] POSTEL, J. *RFC 959 : RFC 959 - File Transfer Protocol* [online]. 1985. 1985 [cit. 2009-03-26]. EN. Dostupný z WWW: <http://www.faqs.org/rfcs/rfc959.html>.

[2] *Wikipedia, The Free Encyclopedia: Windows Mobile.* [online]. [cit. 2009-04-09]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Windows_Mobile>.

[3] *Wikipedia, The Free Encyclopedia: .NET Compact Framework.* [online]. [cit. 2009-04-09]. Dostupný z WWW: <http://en.wikipedia.org/wiki/.NET_Compact_Framework>.

[4] *Wikipedia, The Free Encyclopedia: FTPS.* [online]. [cit. 2009-04-15]. Dostupný z WWW: <http://en.wikipedia.org/wiki/FTPS>.

[5] *Wikipedia, The Free Encyclopedia: File Transfer Protocol.* [online]. [cit. 2009-04-15]. Dostupný z WWW: <http://en.wikipedia.org/wiki/FTP>.

[6] WILTAMUTH , Scott, HEJLSBERG, Anders. *C# Language Specification.* MSDN [online]. 2007 [cit. 2009-04-18]. Dostupný z WWW: <http://download.microsoft.com/download/3/8/8/388e7205-bc10-4226-b2a8-75351c669b09/CSharp Language Specification.doc>

[7] *Wikipedie: Otevřená encyklopedie: C Sharp* [online]. c2009 [citováno 2009-04-18]. Dostupný z WWW: <http://cs.wikipedia.org/w/index.php?title=C_Sharp&oldid=3899513>

[8] *Working with files on Smartphone devices with the .NET Compact Framework.* [online]. Březen 2004 [cit. 2009-04-26]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/aa446567.aspx>

[9] HOROWITZ, M., LUNT, S. *RFC 2228: RFC 2228 - FTP Security Extensions.* [online]. 1997 [cit. 2009-04-28]. EN. Dostupný z WWW: <http://tools.ietf.org/html/rfc2228>.

[10] FORD-HUTCHINSON, P. *RFC 4217: RFC 4217 - Securing FTP with TLS.* [online]. 2005 [cit. 2009-04-30]. EN. Dostupný z WWW: <http://tools.ietf.org/html/rfc4217>

[11] *Wikipedie: Otevřená encyklopedie: File Transfer Protocol* [online]. C2009 [cit. 2009-05-12]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/File_Transfer_Protocol>

[12] *Wikipedia, The Free Encyclopedia: Document Type Definition.* [online]. [cit. 2009-05-12]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Document_Type_Definition>.

[13] *MSDN : Microsoft Developer Network* [online]. 2009. USA : Microsoft, 2009 , 2009 [cit. 2009-05-13]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/>

[14] PECINOVSKÝ, R. *Návrhové vzory : 33 vzorových postupů pro objektové programování.* Odpovědný redaktor: Václav Kadlec. 1. vyd. Brno : Computer Press, 2007. 527 s. ISBN 978-80-251-1582-4.

Seznam příloh

Příloha A. Uživatelský manuál

Příloha B. CD obsahující zdrojové texty aplikace a technické zprávy, aplikaci a instalační balíček

Příloha A

Uživatelský manuál

V tomto uživatelském manuálu jsou naznačeny a vysvětleny možnosti nastavení a ovládání FTP serveru.

Hlavní okno

Hlavní okno programu slouží hlavně pro výpis souhrnných informací a spuštění popřípadě zastavení činnosti serveru. Naleznete zde zejména informace o adrese a portu na kterém server běží, počet aktuálně připojených uživatelů, nebo celkovou dobu běhu serveru. Tyto informace se vždy jednou za deset sekund obnovují.

Obrázek A-1: Hlavní okno při vypnutém serveru

Obrázek A-2: Hlavní okno při zapnutém serveru

Spuštění a zastavení serveru

V hlavním okně se také provádí spuštění a zastavení běhu serveru. K tomuto slouží tlačítko s textem *Offline* (nebo *Online*, jeho text se mění v závislosti na aktuálním stavu serveru). Po jeho stisknutí se spustí (popřípadě zastaví) běh serveru a je možné se k němu připojit na zobrazené adrese a portu.

Nastavení FTP serveru

Nastavení FTP serveru je dostupné přes položky menu *Možnosti* → *Nastavení* v hlavním okně. V tomto okně se dají nastavit parametry serveru. Jedná se o číslo portu, na kterém server očekává klienty, maximální počet souběžně připojených klientů a povolení či zakázání anonymního přístupu.

Obrázek A-3: Dialog nastavení FTP serveru

Nastavení čísla portu

Změna nastavení čísla portu umožňuje provozovat server na jiném než výchozím (21) portu. Po změně portu je nutné aby všichni klienti věděli, že server běží na jiném než výchozím portu. Z tohoto důvodu se nedoporučuje toto nastavení měnit, pokud si nejste jistí výsledkem. Povolený rozsah pro číslo portu je aplikačně omezen na 1 – 65535.

Nastavení maximálního počtu připojených uživatelů

Tato položka nastavení slouží k omezení maximálního počtu souběžně připojených uživatelů. Jejím hlavním účelem je omezení zátěže pro mobilní zařízení. Výchozí hodnotou je omezení na 10 souběžně připojených klientů. Ovšem toto nastavení je plně závislé na výkonu mobilního zařízení a tak je na některých zařízeních možné počet klientů zvýšit, kdežto na některých bude nutné počet snížit. Povolený rozsah hodnot je aplikačně omezen na 1 – 100.

Nastavení anonymního přístupu

Pro povolení anonymního přístupu je třeba zaškrtnout políčko *Povolit anonymní přihlášení*. Jakmile toto provedete můžete zadat cestu pro anonymního uživatele. Cestu je možné zadat ručně, nebo pomocí dialogu *Procházet*.

Pro zakázání anonymního přístupu stačí pouze odškrtnout políčko *Povolit anonymní přihlášení*. Poté již server anonymní uživatele nebude akceptovat.

Obrázek A-4: Zakázání anonymního přístupu

Správa uživatelů

Správa uživatelů je dostupná z hlavního okna přes položky menu *Možnosti* → *Uživatelé*. Okno obsahuje seznam všech nastavených uživatelů a informace o nich a dále možnost přidat, upravit či odstranit uživatele.

Obrázek A-5: Dialog pro správu uživatelů

Přidání nového uživatele

Dialog pro přidání nového uživatele vyvoláte pomocí tlačítka *Přidat* v okně správy uživatelů. Pro úspěšné přidání nového uživatele je třeba vyplnit jedinečné uživatelské jméno, libovolné heslo a přiřadit uživateli existující cestu. V případě nesprávného vyplnění některé z položek Vás aplikace při pokusu o uložení upozorní na chybu.

Obrázek A-6: Dialog pro přidání nového uživatele

Obrázek A-7: Dialog pro upravení uživatele

Upravení existujícího uživatele

Dialog pro upravení již existujícího uživatele vyvoláte výběrem daného uživatele v seznamu a následném stisknutí tlačítka *Upravit*. Tento dialog se příliš neliší od dialogu pro přidání nového uživatele. Jediným rozdílem je nemožnost změny uživatelského jména, všechna ostatní pravidla zůstávají stejná.

Odstranění uživatele

Odstranění uživatele se provádí vybráním daného uživatele v seznamu a následným stisknutím tlačítka *Odebrat*. Aplikace se poté zeptá zda má daného uživatele opravdu odstranit. V případě potvrzení volby uživatele se jeho nastavení odstraní, pokud volbu nepotvrdíte daný uživatel odstraněn nebude.