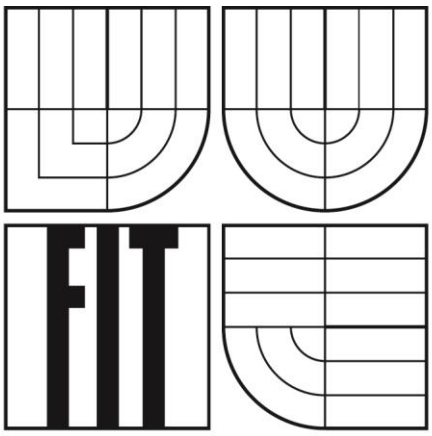


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Síťové aplikace a správa sítí

2015/2016

Projekt:

FTP klient

Peter Miklánek (xmikla10) Brno, 5.11.2016

Obsah

1 Úvod

Úlohou projektu, ktorý popisuje táto dokumentácia, bolo vytvoriť FTP klienta, ktorý vie pracovať v Aktívnom aj Pasívnom móde. Program umožňuje sťahovanie, nahrávanie či vymazávanie súborov zo serveru a výpis adresárovej štruktúry. Rozhodol som sa implementovať aj zopár rozšírení, ktoré budú popísané pri konci tejto dokumentácie.

Testovanie prebiehalo na odporúčanom poskytovanom virtuálnom stroji a taktiež na localhoste a na dostupných serveroch na internete.

2 Úvod do problematiky

Skôr ako začnem popisovať svoje vlastné riešenie, vysvetlím niekoľko termínov, ktoré sú potrebné pre pochopenie danej problematiky projektu.

2.1 File Transfer Protokol

File Transfer Protokol (alebo skrátene FTP) je služba a protokol k prenosu súborov postavených nad TCP.

FTP využíva pre prenos transportný protokol TCP. Vytvára dátové kanály medzi klientom a serverom : jeden pre vlastný prenos na porte 20 a druhý pre prenos príkazov (na porte 21). Pri prenose súborov sa tento kanál využíva pre čítanie súborov aj nahrávanie súborov. Podľa toho, či vytvorenie dátového kanálu iniciuje klient alebo server, rozlišujeme aktívny FTP prenos alebo pasívny FTP prenos.

2.1.1 Komunikácia FTP

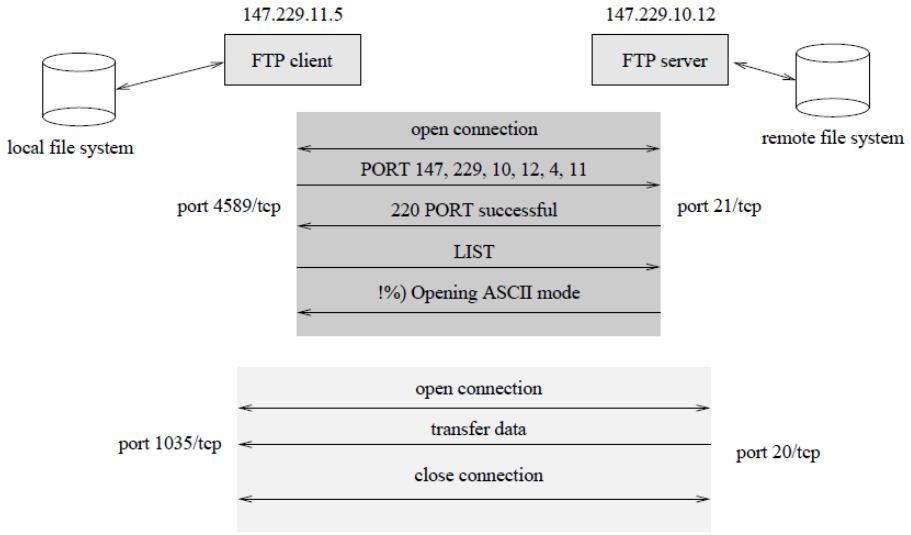
Medzi základné činnosti a príkazy protokolu FTP patrí otvorenie spojenia, autentizácia užívateľa ale hlavne operácie spojené so súbormi a operácie spojené s adresármi.

Tabuľka základných príkazov protokolu FTP :

Príkaz	Popis
USER, PASS	Identifikácia užívateľa, heslo
CWD, PWD	Zmena adresára, tisk aktuálneho adresára
LOGIN, QUIT	Ukončenie prenosu
PORT	V príkaze PORT sa posiela IP adresa a číslo portu
PASV	Pasívny mód
TYPE	Typ kódovania prenášaných dát
RETR	Prenos súborov zo serveru na klienta
STORE, APPEND	Uloženie súborov na server
RMD, MKD	Zrušenie súborov
LIST	Zoznam súborov v adresári

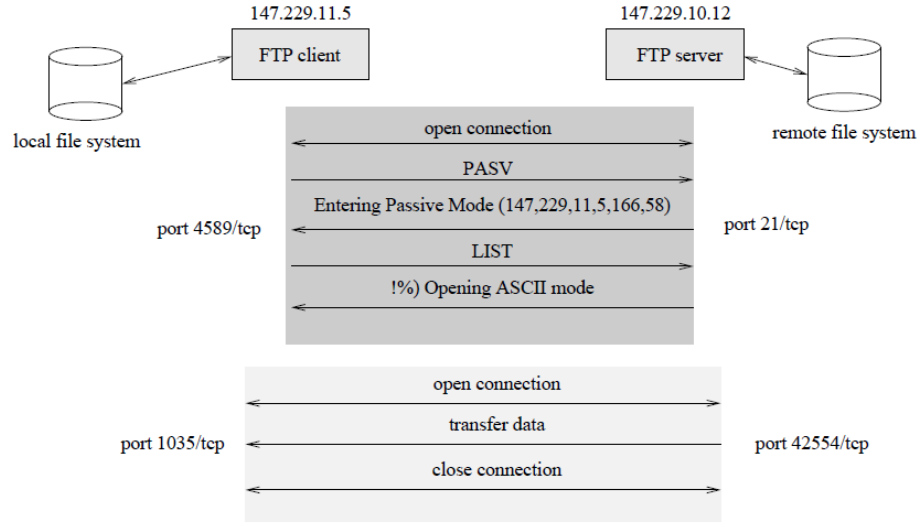
2.1.2 Komunikácia aktívneho prenosu FTP

Komunikácia aktívneho prenosu FTP prebieha tak, že klient inicializuje prenos a vytvorí kanál pre riadenie (pripojí sa na rezervovaný port 21 Ftp serveru). Klient príkazom PORT oznámi svoju IP adresu a číslo dátového portu serveru. Server následne aktívne otvorí dátový kanál zo svojho rezervovaného portu 20 na port klienta, ktorý dostal v príkaze PORT. Následne server pošle klientovi požadované dáta.



2.1.3 Komunikácia pasívneho prenosu FTP

Komunikácia pasívneho prenosu FTP sa požíva napríklad v prípade, keď je FTP klient za NAT serverom. Prebieha tak, že klient inicializuje prenos a vytvorí riadiaci kanál rovnako ako pri aktívnom móde. Klient požiada server o prepnutie do pasívneho režimu príkazom PASV a server mu následne oznámi číslo dátového portu. Klient následne otvorí dátový port pre posielanie a prijímanie dát a server posiela klientovi požadované dáta.



3 Popis vlastnej implementácie a programu

Nasledujúce riadky popisujú moje vlastné riešenie problematiky, návrh riešenia, popis implementácie a popis programu. Hlavnou funkciou celého programu je funkcia *main(),* ktorá slúži pre riadenie celého programu.

3.1 Spracovanie argumentov

Argumenty sa spracovávajú vo funkcií *arguments().* Použil som klasickú metódu spracovania pomocou *int argc* a *char *argv[]* . Pri zadávaní argumentov programu nezáleží na ich poradí, pri malom počte zadaných argumentov sa na štandardný výstup vypíše nápoveda, pri zadaní chybného počtu zadaných argumentov a pri zadaní nesprávneho argumentu sa na štandardný výstup vypíše chyba s krátkym popisom chyby.

3.2 Vytvorenie pripojenia a prihlásenie sa na server

Pre vytvorenie pripojenia a prihlásenia sa na server využívam funkciu *socketFunction().* Na začiatku tejto funkcie používam *socket(),* ktorý mi vytvorí dátovú štruktúru schránky a inicializuje schránku. Následne sa pomocou connect() a getsockname() pripojím k serveru a získam informácie o aktuálnej schránke.

Po pripojení sa na server a získaní všetkých potrebných informácií posielam na server príkazy, pomocou ktorých sa na servery prihlásim. K tomu využívam USER a PASS. Pri prijímaní odpovede od serveru kontrolujem návratový kód. V prípade nesprávneho návratového kódu vypíšem na štandardný výstup chybovú hlášku. Na konci funkcie *socketFunction()* pomocou PWD zisťujem aktuálny adresár, ktorý si ukladám pre neskoršie využitie.

3.3 Aktívny mód

Komunikáciu aktívneho prenosu FTP mi reprezentuje funkcia *activeMode()* . O konkrétnom postupe ako aktívny mód funguje som sa zmienil už vyššie. Podľa tohto postupu som postupoval aj pri implementácií. V prípade, že bol *fclient* spustený bez argumentu reprezentujúceho aktívny alebo pasívny mód, program sám vyberie aktívny mód. Pre správne fungovanie aktívneho módu je veľmi dôležité využitie funkcií *bind(), listen()* a *accept().*

Funkcia *bind()* prepojí schránku s lokálnou adresou a portom zadaným v druhom parametre. Pri mojej implementácií som funkciu *bind()* implementoval do cyklu for so zámerom výberu správneho čísla portu, ktorý je v rozsahu 1025 až 65535.

Funkcia *listen()* nastaví pasívny režim komunikácie pre zadanú nepripojenú schránku.

Funkcia *accept()* vyberie požiadavku z fronty čakajúcich spojení a vytvorí spojenie klient-server.

Pri konci funkcie *activeMode()* zisťujem, či pri spustení *fclienta* bol zadaný argument reprezentujúci sťahovanie, nahrávanie, vymazávanie alebo vytvorenie/vymazanie priečinku. Na základe toho je volaná jedna z týchto funkcií :

Funkcia *get()* reprezentujúca sťahovanie súboru zo server.

Funkcia *put()* reprezentujúca nahrávanie súboru na server.

Funkcia *del()* reprezentujúca vymazanie súboru na servery.

Funkcia *mkd()* reprezentujúca vytvorenie priečinku na servery.

Funkcia *rmd()* reprezentujúca vymazanie priečinku na servery.

Funkcia *nlst()* reprezentujúca výpis súborov v adresári.

V prípade, že bol program spustený bez tejto špecifikácie, je volaná funkcia *ls()* pre výpis adresárovej štruktúry. Na konci sa samozrejme nesmie zabudnúť na zatvorenie už nepotrebných *socketov* pomocou funkcie *close()*.

3.4 Pasívny mód

Komunikáciu pasívneho prenosu FTP mi reprezentuje funkcia *passiveMode()* . O konkrétnom postupe ako pasívny mód funguje som sa taktiež zmienil už vyššie. Podľa tohto postupu som postupoval aj pri implementácií.

Dôležitou časťou tejto funkcie bolo správne spracovanie odpovede od servera, ktorá udáva informácie o IP adrese a čísla reprezentujúce číslo portu, na ktorom očakáva dáta. Pomocou týchto údajov program vypočíta číslo portu.

Pri konci funkcie *passiveMode()* zisťujem, či pri spustení *fclienta* bol zadaný argument reprezentujúci sťahovanie, nahrávanie, vymazávanie alebo vytvorenie/vymazanie priečinku rovnako ako pri funkcií *activeMode()*. Na základe toho je volaná jedna z týchto funkcií :

Funkcia *pasv_get()* reprezentujúca sťahovanie súboru zo server.

Funkcia *pasv_put()* reprezentujúca nahrávanie súboru na server.

Funkcia *pasv_del()* reprezentujúca vymazanie súboru na servery.

Funkcia *mkd()* reprezentujúca vytvorenie priečinku na servery.

Funkcia *rmd()* reprezentujúca vymazanie priečinku na servery.

Funkcia *pasv_nlst()* reprezentujúca výpis súborov v adresári.

V prípade, že bol program spustený bez tejto špecifikácie, je volaná funkcia *pasv_ls()* pre výpis adresárovej štruktúry. Na konci sa taktiež nesmie zabudnúť na zatvorenie už nepotrebných *socketov* pomocou funkcie *close()*.

3.5 Ostatné funkcie

V tejto časti si ešte spomenieme 3 veľmi dôležité funkcie pre správne fungovanie programu *fclient*.

Funkcia *recvM()* slúži pre príjem odpovede zo serveru pomocou *recv()* a následné spracovanie tejto správy. Odpoveď od servera je prijímaná postupne po jednom znaku až pokiaľ sa nepríde na koniec správy. Pri spracovaní správy je zvlášť spracovaných prvých pár znakov pre zistenie trojčíselného kódu, ktorý je využívaný ako kontrola. Touto kontrolou je myslené porovnanie očakávaného kódu s tým prijímaným. Napríklad pri odoslaní príkazu USER je očakávaný kód v odpovedi 331 a v prípade, že tomu tak nie je, program sa ukončí s výpisom chyby na štandardný výstup.

Funkcia *dirStructure()* slúži pre spracovanie adresárovej štruktúry. Táto funkcia sa volá z funkcie *ls()* alebo *pasv_ls()* a spracováva obsah správy z odpovede na príkaz LIST. Funkcia *dirStructure()* prechádza celú odpoveď až do konca po jednom znaku. V odpovedi na príkaz LIST dostávame informácie o aktuálnom adresári a z týchto informácií vie program určiť, či sa jedná o adresár alebo o súbor. Na základe toho, sú priečinky pridávané za seba do reťazca v štruktúre, ktorý slúži ako zásobník. Po spracovaní jedného adresára sa z reťazca vyberá ďalší adresár na spracovanie až pokiaľ nie je reťazec prázdny.

Poslednou z trojice veľmi dôležitých funkcií je funkcia *quit()*. Táto funkcia slúži pre korektné odhlásenie sa zo serveru a zatváranie už nepotrebných socketov pomocou funkcie *close()*.

V programe bolo využitých viacero funkcií, ktoré som tu nerozoberal podrobnejšie a tak sem prikladám aspoň ich výpis a krátky výstižný popis :

Funkcia *print_help()* pre výpis nápovedy.

Funkcia *readFile()* pre čítanie zo súboru.

Funkcia *perror()* pre výpis chybovej hlášky.

Funkcia *editIP()* pre úpravu IP adresy.

Funkcia *cd()* pre zmenu aktuálneho adresára.

Funkcia *selectDir()* ako pomocná funkcia funkcie *dirStructure()*.

4 Návod na použitie

Aby mohol byť program spustený, musí byť najprv preložený. Preloženie programu má na starosti *Makefile* a dá sa spustiť jednoducho príkazom *make*. Následne sa program spúšťa v nasledujúcom tvare :

./fclient -s server -c filename.txt [-p]|[-a port_number] [-d|-u|-r filename] [-P path] [-md|-rd directoryname] [-nl]

-s server IP adresa alebo doménové meno serveru

-c filename.txt súbor s uloženým menom a heslo

-a port_number aktívny mód, dátové spojenie bude inicializované z daného portu

-p pasívny mód

-u filename súbor, ktorý sa uloží na server

-d filename súbor, ktorý sa stiahne na server

-r filename súbor, ktorý sa vymaže zo serveru

-P path cesta k súboru, voliteľný prepínač k -u, -d, -r

-md directoryname adresár, ktorý sa vytvorí na servere

-rd directoryname adresár, ktorý sa vymaže na servere

-nl výpis súborov aktuálneho adresára

5 Rozšírenia

V mojom riešení implementácie programu *fclient* boli implementované 3 rozšírenia : vytvorenie priečinku na servere, vymazanie priečinku na servere a výpis súborov aktuálneho adresára.

Vyššie som už popísal ktoré funkcie tieto činnosti reprezentujú a aj spôsob ich použitia. Základný program FTP klient by podľa mňa mal obsahovať aj

možnosť vytvorenia/vymazania adresára na servere a z tohto dôvodu som sa rozhodol implementovať toto rozšírenie.

Výpis súborov aktuálneho adresára je funkcia reprezentujúca príkaz NLST. Príkaz NLST umožňuje získať prehľadný výpis aktuálnej adresárovej štruktúry. Podobný výpis umožňuje príkaz LIST, ale je potrebné rozsiahlejšie spracovanie aby bol následný výpis prehľadný.

Každé z týchto rozšírení funguje aj pre aktívny mód, pre pasívny mód a taktiež je možné ich kombinovať s prepínačom -P.

6 Záver

Program bol riadne otestovaný na operačnom systéme Ubuntu 16.04, pomocou virtuálneho stroja na VirtualBoxe a taktieť pomocou VSFTPD umiestneného na localhoste Ubuntu.

Program bol tvorený v programovacom jazyku C++ a prekladaný prekladačom g++ pomocou mnou vytvoreného Makefile súboru, ktorý je súčasťou odovzdaného archívu. Behom implementácie programu som pre vizualizáciu správ využíval Wireshark.

Implementácia FTP klienta bola veľmi zaujímavá a aj napriek tomu, že FTP bol vyvinutý už pred viac ako 30 rokmi, myslím si že mi tento projekt poskytol znalosti, ktoré sa mi v budúcnosti určite zídu.

Referencie

[1] W. R. Stevens. TCP/IP Illustated. The Protocols., volume 1. Addison Wesley, 1994.

[4] J.Reynolds and J.Postel, "File Transfer Protocol (TCP)", RFC 959, ISI, October 1985.