

Simple FTP Klient + Server

DOKUMENTÁCIA K PROJEKTU Z IPK

AUTOR PRÁCE

ADAM POLÍČEK

Brno 2022

Obsah

Simple FTP protokol.....	3
RFC 913.....	3
Argumenty projektu.....	3
Implementácia:.....	4
Funkcia concatPath().....	5
USER <user-id>.....	6
PASS <password>.....	7
LIST <F V> <directory-path>.....	8
KILL <file_to_delete>.....	9
Testovanie.....	13
Slovník pojmov.....	13
Použitá literatúra.....	13

Simple FTP protokol

Alebo z angl. Simple File Transfer Protocol je využívaný na prenos konfiguračných dát a súborov medzi klientom a serverom pomocou TCP spojenia.

V porovnaní s TFTP má pokročilé funkcie ako napríklad “directory listing”. Rovnako tak implementuje autorizáciu, kde sa užívateľ musí najprv prihlásiť aby mohol vykonať väčšinu príkazov na serveri. To však nerobí z SFTP bezpečný protokol. Má veľa bezpečnostných slabostí a nemá šifrovanú komunikáciu.

RFC 913

V projekte využívam štandard RFC 913 ustanovený v septembri 1984 Markom K. Lottorom.

· <https://datatracker.ietf.org/doc/html/rfc913> [1]

Argumenty projektu

Pri spúšťaní klienta aj serveru je potreba nastaviť argumenty podľa ktorých sa riadi ďalší chod programu.

Klient:

-h <ip_adresa>

- IP adresa serveru na ktorú sa klient pripojí (ipv6 alebo ipv4).

-p <port>

- Číslo portu pod ktorým server beží.

-f <pracovný_priečinok>

- Absolútna alebo lokálna cesta k pracovnému priečinku klienta.

Server:

-i <rozhranie>

- Rozhranie na ktorom bude server naslúchať (napr. „enp9s0“).

-p <port>

- Port na ktorom server naslúcha.
- Treba pamätať na to že na linuxe sú porty 0-1023 rezervované pre systém.

-f <pracovný_priečinok>

- Absolútna alebo lokálna cesta k pracovnému priečinku serveru.

-u <databáza>

- Absolútna cesta k databáze s databázou užívateľov serveru.

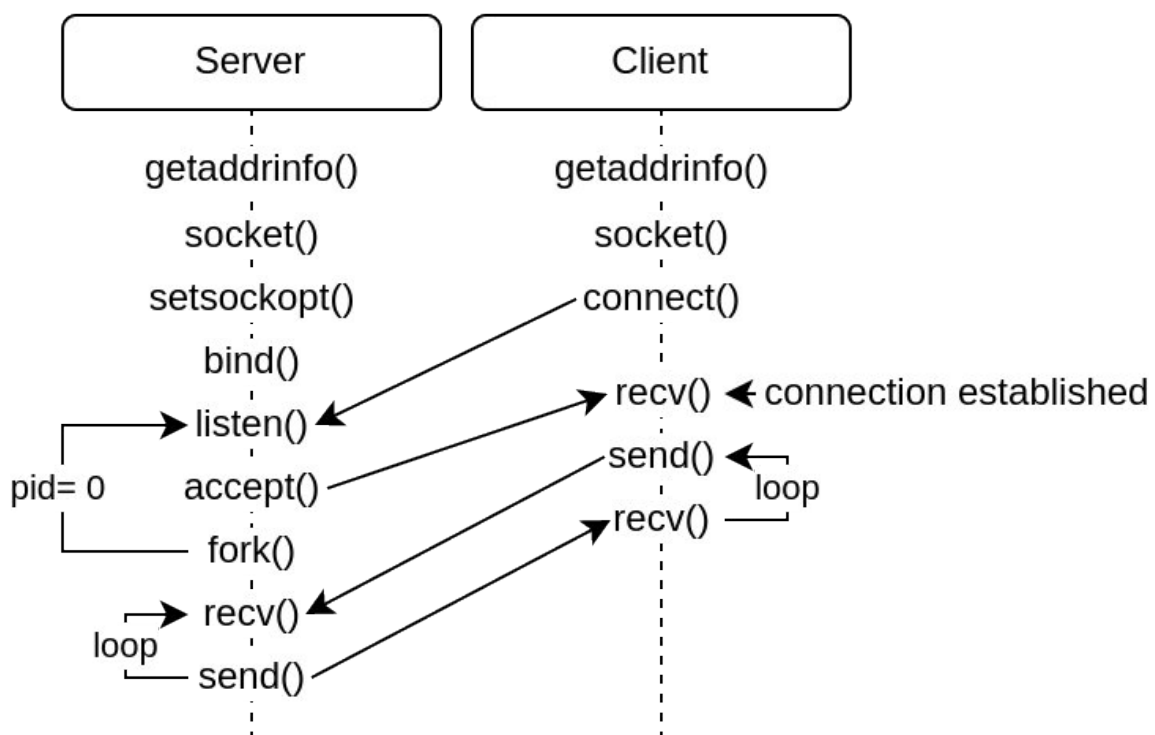
Implementácia:

Projekt podporuje ako ipv6 tak aj ipv4 pomocou hybridného nastavenia socketu. TCP spojenie je nadviazané pomocou knižníc sys/types.h, sys/socket.h a netdb.h.

Kostru projektu som vyhotovil podľa návodu [2] kde je detailne popísané nastavenie socketu pre komunikáciu klient-server, kde nevieme dopredu povedať aká verzia IP bude požadovaná.

```
memset(&hints, 0, sizeof hints); // make sure the struct is empty
hints.ai_family = AF_UNSPEC;      // don't care IPv4 or IPv6
hints.ai_socktype = SOCK_STREAM; // TCP stream sockets
hints.ai_flags = AI_PASSIVE;     // fill in my IP for me
```

Po úspešnom vyhodnotení užívateľských argumentov a vytvorení socketu so správnym nastavením, môže začať komunikácia. Klient sa ihneď pripojí na server definovaný adresou a portom. Server musí bežať a počúvať pomocou funkcie **listen()**. V momente ako klient nadviaže spojenie so serverom, server vytvára nový proces pomocou **fork()**. Hlavný proces stále naslúcha zatiaľ čo potomok zatvára listener a inicializuje komunikáciu s klientom. Lepšie to prezentuje nasledujúci graf vyhotovený podľa mojej implementácie.



Samotná komunikácia klienta a serveru je už jednoducho pomocou funkcií **recv()** a **send()**. Kde klient posieľa žiadosť a server na ňu odpovedá. Validitu príkazov kontroluje hlavne server, klient validuje komplet celý príkaz iba v niektorých prípadoch.

Funkcia concatPath()

Server prijíma všetky cesty k súborom ako lokálne cesty od pracovného priečinku serveru, definovaného argumentom (-f) alebo príkazom od klienta, po požadovaný súbor / priečinok.

Jedine príkaz CDIR neberie lokálnu cestu od definovaného pracovného priečinku, ale od spusteného serveru, na koľko sám tento pracovný priečinok nastavuje.

O prevod lokálnej cesty na absolútnu, spájanie lokálnych ciest a ich validáciu sa stará implementovaná funkcia **concatPath(wd, path, folder)**. Táto funkcia získa aktuálnu cestu k serveru, pridá k nej lokálnu cestu z parametru wd. Potom k nej pridá lokálnu cestu z parametru path, avšak iba ak táto cesta nie je NULL. Celý čas dbá na to aby boli cesty správne naformátované a teda aby dokázala poskladať validnú cestu z ľubovoľne zadaného vstupu. Na koniec skontroluje či sa na vytvorenej ceste nachádza priečinok (folder= 1), alebo súbor (folder= 0). Ak sa nachádza tak vráti poskladanú absolútnu cestu k nemu. Ak nie tak vráti „error“. Takto vytvorenú absolútnu cestu využívajú príkazy LIST, KILL, atď. Samotná lokálna cesta od Serveru po jeho pracovný priečinok je vždy držaná v perzistentnej premennej workingDir.

Klient má tiež implementovanú obdobu tejto funkcie ale je podstatne zjednodušená a nevyužíva sa tak často.

Príkazy:

Užívateľ na strane klienta zadáva do terminálu príkaz vo forme:

- <príkaz> [argumenty]

jednotlivé príkazy podľa štandardu RFC 913 sú nasledovné:

- USER, ACCT, PASS, TYPE, LIST, CDIR, KILL, NAME, DONE, RETR, STOR

Následne server príkaz validuje a prevedie požadovanú činnosť. Podľa toho vracia odpoveď na ktorej začiatku je vždy jeden z nasledujúcich response code:

- +, -, !

Príkazy nemusia byť zadané vo formáte uppercase, na koľko ich ako server, tak aj klient porovnávajú funkciou **strncasecmp()** / **strncasecmp()**, ktorá berie uppercase ako aj lowercase písmená rovnako.

USER <user-id>

Pri spúšťaní serveru užívateľ definuje absolútnu cestu k databáze s užívateľmi v argumente **-u <abs_path>**. Pre prehľadávanie tejto databázy som implementoval funkciu **getUser(username, filePath, password)**. Pri tomto príkaze je posledný parameter funkcie NULL, keďže heslo ešte nemáme k dispozícii. Táto funkcia sa pokúsi nájsť užívateľské meno v databáze a podľa toho vracia správu pre klienta.

-unable to access database, server error

- Toto je chyba serveru, ktorá značí že sa nepodarilo otvoriť databázu.

+User-id valid, send password

- Username je nájdené v databáze. Pre prihlásenie zostáva už len poslať heslo.
- Pri tejto odpovedi sa zadané username uloží do perzistentnej premennej.

-Invalid user-id, try again

- Username sa nenašlo v databáze. Užívateľ môže poslať príkaz znovu, pravdepodobne už s iným username.

ACCT <account>

Server nepodporuje spoplatnenie užívateľov. Tento príkaz tak nemá využitie a užívatelia sú identifikovaný podľa ich užívateľského mena zadaného v príkaze USER.

! Account valid, logged-in

- Ak je už užívateľ prihlásený tak server vráti túto odpoveď.

-there is no need for ACCT, just login

- Odpoveď serveru v ostatných prípadoch.

PASS <password>

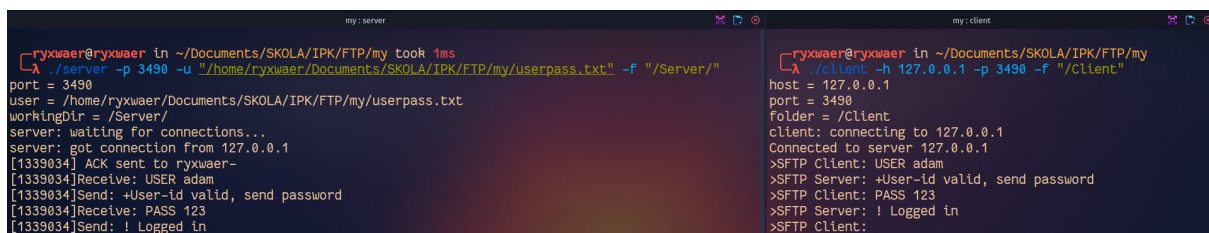
Tak isto ako USER, aj tento príkaz využíva funkcie getUser(username, filePath, password). Tento sú vyplnené všetky argumenty funkcie. Username je uložené z predošlého volania príkazu USER. filePath je cesta k databáze užívateľov. Password je hľadané heslo.

-Wrong password, try again

- zadané heslo sa nezhoduje s heslom daného užívateľa v databáze.

! Logged in

- Heslo sa zhoduje s heslom v databáze
- perzistentná premenná **loginVerification** sa nastaví na **1**. To znamená, že užívateľ získava prístup ku všetkým nasledujúcim príkazom požadujúcim autorizáciu.



```
my: server
ryxwaer@ryxwaer in ~/Documents/SKOLA/IPK/FTP/my took 1ms
λ ./server -p 3490 -u "/home/ryxwaer/Documents/SKOLA/IPK/FTP/my/userpass.txt" -f "/Server/"
port = 3490
user = /home/ryxwaer/Documents/SKOLA/IPK/FTP/my/userpass.txt
workingDir = /Server/
server: waiting for connections...
server: got connection from 127.0.0.1
[1339034] ACK sent to ryxwaer-
[1339034]Receive: USER adam
[1339034]Send: +User-Id valid, send password
[1339034]Receive: PASS 123
[1339034]Send: ! Logged in

my: client
ryxwaer@ryxwaer in ~/Documents/SKOLA/IPK/FTP/my
λ ./client -h 127.0.0.1 -p 3490 -f "/Client"
host = 127.0.0.1
port = 3490
folder = /Client
client: connecting to 127.0.0.1
Connected to server 127.0.0.1
>SFTP Client: USER adam
>SFTP Server: +User-Id valid, send password
>SFTP Client: PASS 123
>SFTP Server: ! Logged in
>SFTP Client:
```

TYPE < A | B | C >

Type súži pre kompatibilitu, na príklad so strojmi kde sú 9-bitové byty. Reálne tento príkaz nemá žiaden efekt. Na výber je z troch typov – ASCII, BINARY a CONTINUOUS.

+Using { Ascii | Binary | Continuous } mode

- Typ bol úspešne nastavený.

-Type not valid

- Nebol zadáný známy typ.

-login needed for TYPE command

- Pre tento príkaz je požadované prihlásenie užívateľa.

LIST <F | V> <directory-path>

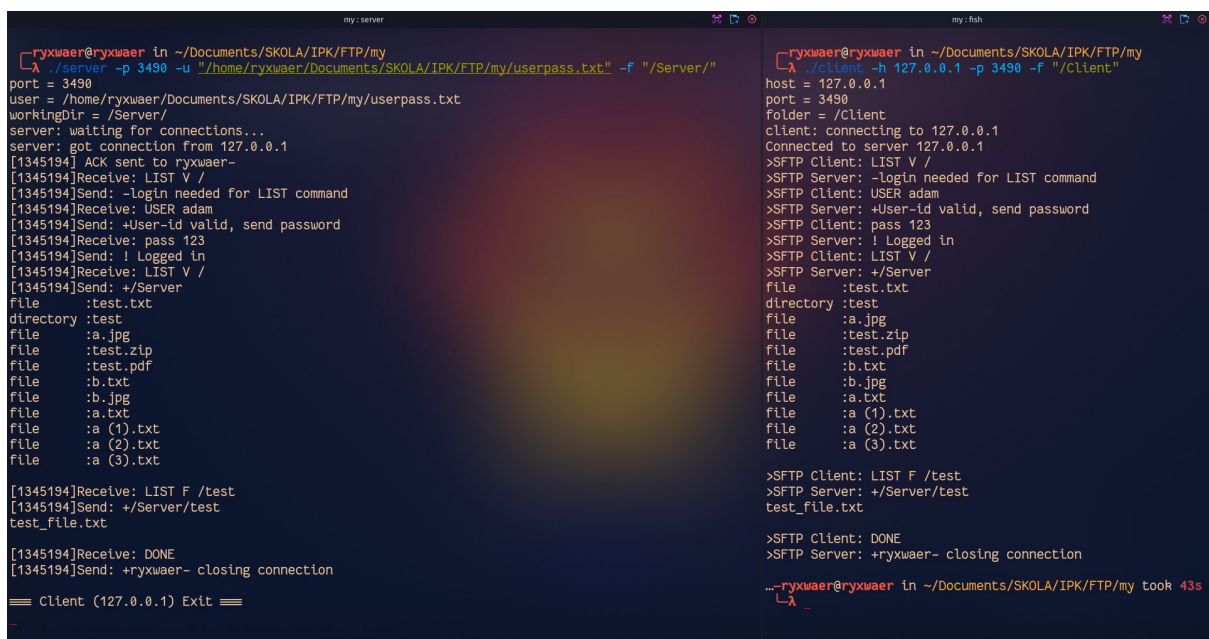
Parameter F alebo V určuje či chceme odpoveď ako jednoduchý zoznam súborov (f) alebo chceme verbose list, ktorý obsahuje informácie na viac (v). Verbose list pridáva pred každý názov súboru, jeho typ: **directory**, **file**, **unknown**. Pre tento príkaz server využíva implementovanú funkciu **getList(path, verbose)**. Path je absolútna cesta k priečinku z ktorého chceme získať výpis súborov. Táto cesta je získaná pomocou funkcie **concatPath()**.

-file not found, try different path

- Zadaná cesta neexistuje.

-wrong LIST option

- Zadaný iný parameter ak V / F alebo nebol zadaný žiaden.



```
my: server
ryxwaer@ryxwaer in ~/Documents/SKOLA/IPK/FTP/my
λ ./server -p 3490 -u "/home/ryxwaer/Documents/SKOLA/IPK/FTP/my/userpass.txt" -f "/Server/"
port = 3490
user = /home/ryxwaer/Documents/SKOLA/IPK/FTP/my/userpass.txt
workingDir = /Server/
server: waiting for connections...
server: got connection from 127.0.0.1
[1345194] ACK sent to ryxwaer-
[1345194]Receive: LIST V /
[1345194]Send: -login needed for LIST command
[1345194]Receive: USER adam
[1345194]Send: +User-id valid, send password
[1345194]Receive: pass 123
[1345194]Send: ! Logged in
[1345194]Receive: LIST V /
[1345194]Send: +/Server
file      :test.txt
directory :test
file      :a.jpg
file      :test.zip
file      :test.pdf
file      :b.txt
file      :b.jpg
file      :a.txt
file      :a (1).txt
file      :a (2).txt
file      :a (3).txt
[1345194]Receive: LIST F /test
[1345194]Send: +/Server/test
test_file.txt
[1345194]Receive: DONE
[1345194]Send: +ryxwaer- closing connection
=== Client (127.0.0.1) Exit ===

my: fish
ryxwaer@ryxwaer in ~/Documents/SKOLA/IPK/FTP/my
λ ./client -h 127.0.0.1 -p 3490 -f "/Client"
host = 127.0.0.1
port = 3490
folder = /Client
client: connecting to 127.0.0.1
Connected to server 127.0.0.1
>SFTP Client: LIST V /
>SFTP Server: -login needed for LIST command
>SFTP Client: USER adam
>SFTP Server: +User-id valid, send password
>SFTP Client: pass 123
>SFTP Server: ! Logged in
>SFTP Client: LIST V /
>SFTP Server: +/Server
file      :test.txt
directory :test
file      :a.jpg
file      :test.zip
file      :test.pdf
file      :b.txt
file      :b.jpg
file      :a.txt
file      :a (1).txt
file      :a (2).txt
file      :a (3).txt
>SFTP Client: LIST F /test
>SFTP Server: +/Server/test
test_file.txt
>SFTP Client: DONE
>SFTP Server: +ryxwaer- closing connection
...ryxwaer@ryxwaer in ~/Documents/SKOLA/IPK/FTP/my took 43s
λ
```

- Na screenshotsote môžeme vidieť: vyžadované prihlásenie užívateľa, lokálnu cestu k priečinku „/“ a „/test“, výpis nájdených súborov ako verbose a basic.

CDIR <new-directory>

Tento príkaz mení pracovný priečinok serveru. Jednoducho sa zavolá **concatPath(new_directory, NULL, 1)** a ak nevráti error tak sa perzistentná premenná **workingDir** nastaví na lokálnu cestu z požiadavku od klienta.

-Can't connect to directory

- zadaná cesta k požadovanému priečinku nie je validná.

!Changed working dir to <working_dir>

- Cesta do pracovného priečinku serveru úspešne zmenená.

KILL <file_to_delete>

Vymaže zadaný súbor. Zavolá funkciu `concatPath(workingDir, file_to_delete, 0)` a ak sa vráti absolútna cesta k tomuto súboru tak ho vymaže pomocou funkcie **`remove()`**.

+<file_to_delete> deleted

- Súbor bol úspešne vymazaný

-wrong file path, -file not found, -unable to delete file, -login first

- chybové hlásenia, tu nedochádza k vymazaniu súboru

NAME <old-file>

Tento príkaz skontroluje dostupnosť zadaného <old-file> pomocou `concatPath(...)` funkcie a ak súbor existuje tak si jeho absolútnu cestu uloží do persistentnej premennej `fileToRename`.

Následne hocikedy sa užívateľ rozhodne súbor premenovať, stačí keď v rámci tejto session zašle príkaz **TOBE <new-name>**. Tento príkaz premenuje súbor na nové, zadané meno.

Príkaz TOBE na rozdiel od NAME pracuje iba s menom súboru, nie s lokálnou cestou k nemu.

+File exists

- Súbor existuje, je uložený do premennej a po príkaze TOBE bude premenovaný.

-wrong file name

- Názov / cesta súboru nebola zadaná = „“ / NULL / „/“

-Can't find <old-file>

- Súbor neexistuje, prípadné zaslanie TOBE nebude mať žiaden efekt.

+<old-file> renamed to <new-name>

- Súbor s lokálnou cestou z pracovného súboru serveru = <old-file> bol úspešne premenovaný na <new-name>.

-unable to rename file

- Chyba oprávnení pravdepodobne.

-send NAME first

- Túto hlášku server vráti ak je zadaný príkaz TOBE skôr ako NAME.

```

[1350324]Send: ! Logged in
[1350324]Receive: NAME /test/test_file.txt
concatPath: /home/ryxwaer/Documents/SKOLA/IPK/FTP/my/Server/test/test_file.txt
[1350324]Send: +File exists
[1350324]Receive: TOBE renamed_file.txt
rename /home/ryxwaer/Documents/SKOLA/IPK/FTP/my/Server/test/test_file.txt to /home/ryxwaer/Documen
ts/SKOLA/IPK/FTP/my/Server/test/renamed_file.txt
[1350324]Send: +test_file.txt renamed to renamed_file.txt
[1350324]Receive: LIST V /test
concatPath: /home/ryxwaer/Documents/SKOLA/IPK/FTP/my/Server/test
[1350324]Send: +/Server/test
file :renamed_file.txt

[1350324]Receive: KILL /test/renamed_file.txt
concatPath: /home/ryxwaer/Documents/SKOLA/IPK/FTP/my/Server/test/renamed_file.txt
[1350324]Send: +/test/renamed_file.txt deleted
[1350324]Receive: LIST V /test
concatPath: /home/ryxwaer/Documents/SKOLA/IPK/FTP/my/Server/test

>SFTP Client: user adam
>SFTP Server: +User-id valid, send password
>SFTP Client: pass 123
>SFTP Server: ! Logged in
>SFTP Client: NAME /test/test_file.txt
>SFTP Server: +File exists
>SFTP Client: TOBE renamed_file.txt
>SFTP Server: +test_file.txt renamed to renamed_file.txt
>SFTP Client: LIST V /test
>SFTP Server: +/Server/test
file :renamed_file.txt

>SFTP Client: KILL /test/renamed_file.txt
>SFTP Server: +/test/renamed_file.txt deleted
>SFTP Client: LIST V /test
>SFTP Server: +/Server/test

>SFTP Client:

```

- Na screenshots môžeme vidieť premenovanie súboru v zanorenom priečinku a jeho následné vymazanie. Tak isto by šlo zmeniť pracovný priečinok na „test“ pomocou príkazu CDIR a potom zadávať už len názov súboru na miesto lokálnej cesty k nemu. Voľba je na užívateľovi, snažil som sa podporiť všetky prístupy.
- Je potrebné si všimnúť že príkaz NAME môže obsahovať lokálnu cestu k súboru ale príkaz TOBE už musí obsahovať iba jeho meno, nezáleží na tom kde sa súbor nachádza.

DONE

Príkaz slúži na ukončenie spojenia medzi serverom a klientom. Klient sa odpojí a skončí. Server ukončí proces, ktorý obsluhoval klienta ale svoju činnosť neukončuje. Zostáva obsluhovať iných užívateľov a naslúchať na prichádzajúce spojenia.

+<host> closing connection

- Potvrdenie ukončenia spojenia pre klienta.

RETR <file-path>

Server zavolá funkciu `concatPath(workingDir, file_path, 0)` aby zistil či súbor na zadanej lokálnej ceste existuje. Ak existuje tak si jeho absolútnu cestu uloží do perzistentnej premennej a odošle klientovi odpoveď obsahujúcu počet bytov súboru. Ako náhle klient obdrží pozitívnu odpoveď (+) od serveru tak automaticky skontroluje či má dostatok miesta pre súbor a ak áno, tak serveru odošle požiadavku **SEND**. Inak by odoslal **STOP**. Dostupné miesto si klient kontroluje vo funkcií **GetAvailableSpace(path)**. Ak server obdrží príkaz STOP tak zašle pozitívnu odpoveď (+ok, RETR aborted) a o nič ďalšie sa nepokúša. Ak obdrží SEND tak súbor otvorí a posiela jeho bity klientovi pomocou funkcie `write(...)`. Ak je súbor väčší ako definovaná konštanta `MAXDATASIZE`, tak sa posiela vo while loope až kým nie je odoslaný celý. Klient ho rovnako tak prijíma pomocou while loopu a ukladá pod rovnakým menom vo svojom pracovnom priečinku.

-wrong file name

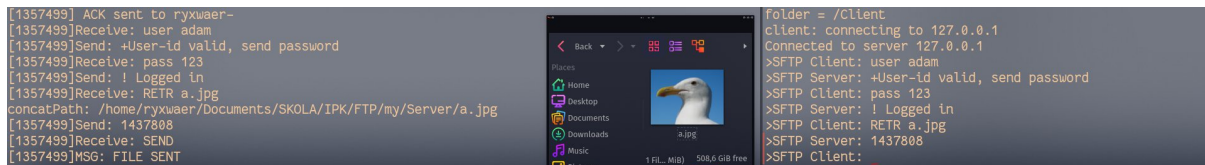
- Chybne zadané meno súboru („ / „ / NULL).

-File doesn't exist

- Nepodarilo sa nájsť zadaný súbor.

-send RETR command first

- Ak server obdrží SEND skôr ako RETR. Tento scenár sa pri mojej implementácii klienta nestane, nakoľko klient posiela SEND automaticky po RETR.

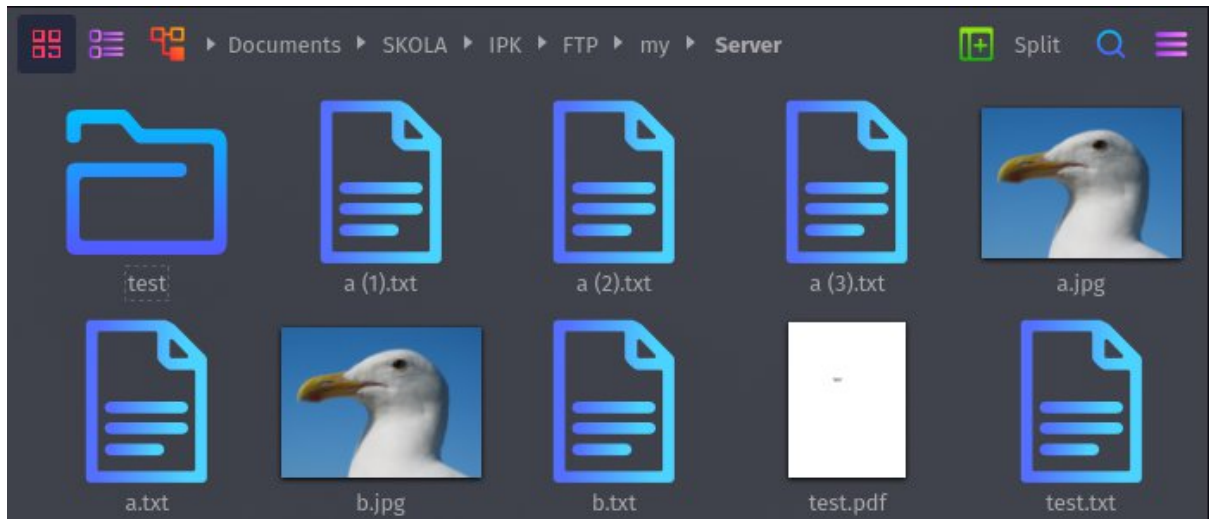


- Po zadaní RETR a.jpg sa vyhodnotí že klient má dostatok miesta na prijatie tohto obrázku a prepošle sa v niekoľkých iteráciách z pracovného priečinku serveru do pracovného priečinku klienta.

STOR < NEW | OLD | APP > <file-path>

Tento príkaz je opak príkazu RETR s rozšírením o kontrolu spôsobu zápisu. To je dosť podstatné, na koľko na server máme zložitejší prístup ako na klienta. Toto riešia zadané parametre NEW, OLD alebo APP. Ak v pracovnom priečinku serveru neexistuje súbor s rovnakým menom ako prijímaný súbor, tak server vytvorí nový súbor s týmto menom do ktorého nahrá prijaté dáta. Ak však súbor s rovnakým menom už existuje tak to ako sa server zachová definujú nasledujúce parametre:

- NEW – súbor zaslaný na server je vytvorený s novým menom. Na koniec starého mena je pridané číslo v zátvorkách od 1 pravdepodobne po maximum integeru.



- OLD – starý súbor s rovnakým menom je prepísaný. Toto je základná funkcionality keď sa súbor otvorí ako „w“ ako write.
- APP - nový súbor je pridaný na koniec starého. Pre dosiahnutie tejto funkcionality sa musí nahráť 1 do perzistentnej premennej a neskôr sa vďaka podmienke kontrolujúcej túto premennú otvorí daný súbor v režime „a“ ako append. Toto nastavenie nefunguje na obrázkoch alebo iných zložitých formátoch.

Testovanie

Projekt bol testovaný na localhoste. Každý implementovaný príkaz som ručne pretestoval ako na validné zadanie tak aj nevalidné. Príkazy STOR a RETR boli testované na textových súboroch, jpg obrázkoch a pdf dokumentoch o maximálnej veľkosti 1,4MiB avšak mali by zvládnuť oveľa väčšiu veľkosť. Tieto testy prebehli všetky bez problémov a projekt som bol schopný využiť na rozsiahle prechádzanie zložkami, listovanie ich obsahu a operácie so súbormi na rôznych úrovniach zanorenia.

Slovník pojmov

Lokálna cesta

- Cesta od aktuálneho pracovného priečinku serveru po požadovaný súbor / priečinok.
- pracovný priečinok sa nastavuje buď argumentom pri spúšťaní serveru (-f) alebo príkazom CDIR zo strany klienta
- CDIR a (-f) berie lokálnu cestu od priečinku so spusteným serverom po požadovaný súbor / priečinok.

Perzistentná premenná

- premenná, ktorá je definovaná v serveri nad hlavným while loopom. Tým pádom jej hodnota pretrváva naprieč všetkými dotazmi klienta pokým server nedostane dotaz, ktorý túto premennú vymaže alebo dokým sa klient neodpojí.

Použitá literatúra

RFC 913: Simple file transport protocol standard [online]. September 1984 [cit. 2022-03-13].

Dostupné z: <https://datatracker.ietf.org/doc/html/rfc913> [1]

Beej's Guide to Network Programming [online]. November 20, 2020 [cit. 2022-03-13].

Dostupné z: <https://beej.us/guide/bgnet/html/#getaddrinfo-prepare-to-launch> [2]