
# Projektová dokumentácia

**Počítačové komunikácie a siete (IPK)**

Simple File Transfer Protocol

10.04.2022 Andrej Hýroš (xhyros00)

# Obsah

# 1 Úvod

Cieľom projektu bolo vytvoriť dva programy (server a klient) implementujúce Simple File Transfer Protocol [4]. Klient sa na základe poskytnutých argumentov pripojí cez BSD schránku [5] na server, kde spustený server naslúcha a čaká na pripojenia. Klient následne posiela dotazy na server, ktorý tieto dotazy spracuváva a adekvátne klientovy odpovedá.

# 2 Simple File Transfer Protocol

Jedná sa o jednoduchú, nezabezpečenú verziu protocolu FTP [1]. Pracuje nad TCP [6] na porte číslo 115. SFTP obsahuje 11 príkazov realizujúcich autentifikáciu užívateľa, prechádzanie adresárovou štruktúrou, presúvanie a premenovávanie a mazanie súborov. Je definovaný dokumentom RFC 913 [4].

## 2.1 Príkazy

SFTP príkazy sa vždy skladajú z štyroch ASCII písmen (akejkoľvek velkosti), následuje medzera a potom argumenty oddelené medzerami. Zoznam príkazov (<arg> vyjadruje povinny argument, {arg | arg} vyjadruje množinu povinných argumentov s pevne danými hodnotami):

- `USER <userid>`
- `ACCT <account>`
- `PASS <heslo>`
- `TYPE { A | B | C }`
- `LIST { F | V } directory-path`
- `CDIR <new-directory>`
- `KILL <file-spec>`
- `NAME <spec>`
- `DONE`
- `RETR <file-spec>`
- `STOR  NEW | OLD | APP  <file-spec>`

Server na príkazy odpovedá vo formáte: "`<response_code> <msg>`"
kde `response_code` je znak z množiny { + | - | ! }, vyjadrujúci výsledok dotazu.

- \+ Úspech
- \- Neúspech
- ! Prihlásený

Následujúce riadky popisujú príkazy tak, ako sú implemetované v tomto riešení projektu.

### 2.1.1 USER

syntax: `USER userid`
userid je užívateľské meno na servery. Po zadaní userid môžu od serveru prísť tri rôzne odpovede:
`+<userid> valid, send password` - userid je validné
`-Invalid user-id, try again` - userid nie je validné
`!<userid> logged in` - heslo už bolo poskytnuté a userid je validné

### 2.1.2 ACCT

V kontexte nášho zadania projektu nemá tento príkaz význam, pre úplnosť je ale implementovaný s rovnakou funkcionalitou ako príkaz USER.

### 2.1.3 PASS

syntax: `PASS <heslo>`
heslo je heslo užívateľa na servery. Odpovede od serveru môžu byť:
`+<heslo> valid, send userid` - heslo je validné
`-Wrong password, try again` - heslo nie je validné
`!<userid> logged in` - userid už bolo poskytnuté a heslo je validné

Poznámka ku autentizácii:
Na zadávanie príkazov iných ako USER/ACCT/PASS je potrebné sa prihlásiť. Užívateľ sa môže príhlásiť zadaním userid a následne príslušného hesla. Tento proces ale funguje aj opačne, a je možné najprv zaslať heslo a následne k nemu príslušné userid. Ak užívateľ začne proces prihlasovania napríklad príkazom USER s argumentom `userid_A`, server bude očakávať heslo `heslo_A`. Ak si užívateľ rozmyslí, že sa chce prihlásiť pod iným userid `userid_B`, musí to oznámiť serveru novým dotazom USER s argumentom `userid_B`. Toto platí aj pri opačnom postupe PASS -> USER/ACCT.

### 2.1.4 TYPE

syntax: `TYPE { A | B | C }`
Tento príkaz je implementovaný pre úplnosť a nemá vpliv na spôsob prenosu súborov. Od serveru môžu prísť odpovede:
`+Using { Ascii | Binary | Continuous } mode`
`-Type not valid`

### 2.1.5 LIST

syntax: `LIST { F | V } <directory-path>`
Príkaz vypíše obsah adresára directory-path. directory-path je volitelný argument a ak nie je zadaný, preskúma sa momentálny pracovný adresár. Pri zadaní možnosti V (verbose) sa vypíšu extra informácie o súboroch ako napríklad veľkosť súboru. Pri zadaní možnosti F vypíše obsah adresára bez extra informácií. Odpovede od serveru môžu byť:
`+<directory-path>:` a následne každá položka adresára na nový riadok.
`-Folder does not exist, try again...` - v prípade že špecifikovaný adresár neexistuje.

### 2.1.6 CDIR

syntax: `CDIR <new-directory>`
Príkaz slúži na zmenu momentálneho pracovného adresára. Argument new-directory je povinný a špecifikuje

adresár, do ktorého sa chce užívateľ presunúť. Odpovede od serveru môžu byť:

`-Can't connect to directory because: does not exist` - v prípade, že špecifikovaný adresár neexistuje.

`-Can't connect to directory because: is not directory` - v prípade, že špecifikovaná cesta nie je adresár.

`+Changed directory to <new-directory>` - v prípade úspechu.

Príkaz podporuje absolútne aj relatívne cesty.

### 2.1.7 KILL

syntax: `KILL <spec>`
Príkaz na mazanie súborov a adresárov. Argument spec je povinný a špecifikuje adresár/súbor ktorý bude odstánený. Odpovede od serveru môžu byť:

`+<spec> deleted` - v prípade úspešného odstránenia.

`-Not deleted (error message)` - v prípade neúspechu, error message je chybová správa od systému.

### 2.1.8 NAME

syntax: `NAME <spec>`
Príkaz slúži na premenovanie súboru/adresára. Tento príkaz sa skladá z dvoch po sebe zaslaných príkazov NAME a následne TOBE, ktorý špecifikuje nový názov. Odpovede od serveru môžu byť:

`+File exists` - ak takýto súbor/adresár existuje.

`-Can't find <spec>` - ak takýto súbor/adresár nebol nájdený.

V prípade úspechu úžívatel následne zašle dotaz `TOBE <new-name>`, kde new-name špecifikuje nový názov súboru/adresára. Odpovede od serveru na tento dotaz môžu byť:

`-Send NAME query first` - ak predchádzajúci dotaz nebol NAME.

`-Failed to rename file, reason: name already taken` - v prípade že súbor/adresár s takýmto menom už existuje. Užívateľ môže zaslať nový dotaz TOBE s iným argumentom.

`-Failed to rename file, reason: unknown` - Premenovanie zlyhalo z neznámeho dôvodu.

`+<spec> renamed to <new-name>` - v prípade úspechu.

### 2.1.9 DONE

syntax: `DONE`
Tento príkaz nemá žiadne argumenty. Klient ním oznamuje serveru, že ukončuje spojenie.

### 2.1.10 RETR

syntax: `RETR <file-spec>`
Tento príkaz oznámi serveru, že klient má záujem o stiahnutie súboru zo serveru. file-spec je povinný argument a špecifikuje súbor určený na prenos. Tento príkaz sa opäť skladá z dvoch dotazov: RETR a následne SEND/STOP. Odpovede od serveru môžu byť:

`-File does not exist` - v prípade, že špecifikovaný súbor neexistuje.

`<file-size>` - v prípade, že súbor existuje, server odošle veľkosť daného súboru v bajtoch ako ASCII znaky.

Klient následne zašle jeden z dotazov STOP alebo SEND, kde odpovede na STOP môžu byť:

`-No RETR query to be stopped` - klient naposlal dotaz RETR.

`+ok, RETR aborted` - príprava prenosu bola úspešne ukončená.

a na dotaz SEND server klientovy odošle daný súbor a následnú odpoveď: `+File send successfully`
Ak klient pošle dotaz SEND bez predchádzajúceho RETR, server odpovie: `-Send RETR query first`

#### 2.1.11 STOR

syntax: `STOR NEW | OLD | APP <file-spec>`
Oznámi serveru, že klient chce na server poslať súbor file-spec. file-spec je povinný argument. Možnosť NEW špecifikuje, že ak súbor s daným menom na servery už existuje, tak sa nemá prepísať. Súbor bude potom na servery uložený pod menom file-spec-copy.format. Odpovede od serveru:
`+File exists, will create file with name file-spec-copy` - súbor existuje, k menu ukladaného súboru bude pripojená koncovka "copy".
`+File does not exist, will create new file` - server je pripravený prijať súbor file-spec.

Možnosť OLD špecifikuje, že ak už súbor s daným menom na servery existuje, má byť prepísaný novým. Odpovede od serveru:
`+Will write over old file` - daný súbor už existuje a bude prepísaný. Server je pripravený prijať súbor.
`+Will create new file` - súbor na servery ešte neexistue a server je pripravený prijať ho.

Možnosť APP nie je v tejto implementácii podporovaná. Server na takýto dotaz odpovie:
`-Appending not supported`
V prípade pozitívnej odpovede klient zašle dotaz `SIZE <filesize>`, špecifikújúci velkosť prenášaého súboru v bajtoch. Odpovede od serveru:
`-Send STOR query first` - Predchádzajúci dotaz nebol STOR.
`-Not a valid filesize` - V prípade že klientom zaslaná velkosť súboru nie je validná. (napríklad obsahuje nenumerické znaky)
`+Saved <filename>` - Súbor bol na servery úspešne uložený.

## 3 Návrh a popis implementácie

Aplikácia je napísaná v jazyku C++ a organizovaná do niekoľkých zdrojových a hlavičkových súborov, ktoré implementujú špecifické časti programu. Pre prácu so súborovým systémom bola použitá knižnica Filesystem [2] (`std::filesystem`), ktorá bola do štandartnej knižnice jazyka zahrnutá vo verzii C++17.

### 3.1 Vstupné body programu, `main.cpp` a `mainc.cpp`

#### 3.1.1 Server

Vstupným bodom serveru je súbor `main.cpp`. Program spracuje vstupné argumenty a spustí server.

#### 3.1.2 Klient

Vstupným bodom klienta je súbor `mainc.cpp`. Program spracuje vstupné argumenty a spustí klienta.

### 3.2 Spracovanie argumentov, `argparser.cpp/h` a `argparser_client.cpp/h`

Tieto súbory implementujú triedy realizujúce parsovanie argumentov. Každá z nich je v súbore `main.cpp`, respektíve `mainc.cpp` inštanciovaná práve raz. Odkaz na túto inštanciu je potom predaný serveru/klientovy,

ktorý si z nej bude ťahať potrebné informácie. V prípade neplatných argumentov je program ukončený s chybovým kódom 4.

### 3.2.1 Chybové kódy

Aplikácia využíva 4 chybové kódy približujúce, k akej chybe v prípade spadnutia aplikácie došlo. Výpis ich slovných názvov s číselnou hodnotou:

- `CONNECTION_ERROR 1` - Chyba pri nadväzovaní spojenia
- `FILE_IO_ERROR 2` - Chyba pri čítaní/zápise do súboru
- `TRANSMISSION_ERROR 3` - Chyba pri prenose súbore
- `CMD_ARGUMENT_ERROR 4` - Chybný vstupný argument

## 3.3 Spracovanie chýb, `error.cpp/h`

V týchto súboroch je definovaná funkcia `error_call` a makrá pre výpis chýb na štandartný chýbový výstup. Táto funkcia je volaná pri výskyte chyby v programe a korektne ho ukončuje.

## 3.4 Pomocné funkcie, `utils.cpp/h`

Obsahuje rôzne pomocné fukncie, ktoré sú všeobecne použitelné a preto nepatria do vlastnej špecifickej triedy.

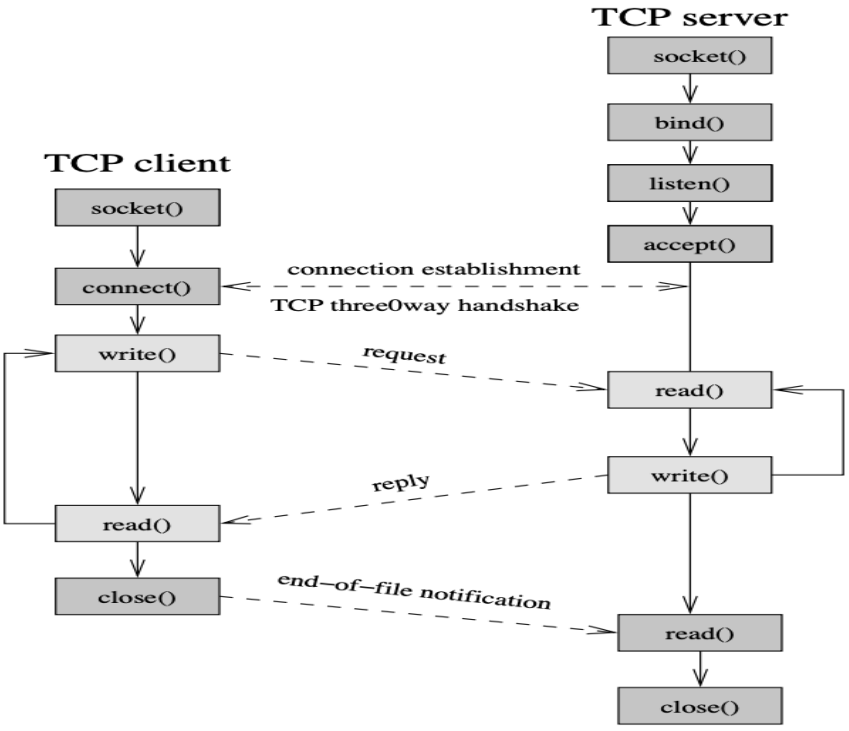
## 3.5 Server, `server.cpp/h`

Obsahuje triedu reprezentujúcu server a implementuje všetkú funkcionalitu serveru. Činnosť serveru v bodoch:

- 1. Spustenie naslúchnia
- 2. Čakanie na spojenie s klientom
- 3. Komunikácia s klientom
- 4. Ukončenie spojenia s klietom a návrat na bod 2

### 3.5.1 Spustenie naslúchnia a spojenie s klientom

Naviazanie spojenia prebieha štandartným spôsobom (viď diagram nižšie) s využitím sieťových hlavičkových súborov `socket.h, netdb.h` [5] atď...
Server podporuje pripojenie cez IPv4 aj IPv6 a využíva protokol TCP [6]. Ak nebol pri spúštaní špecifikovaný port, bude naslúchať na defaultnom porte 115. Naslúcha na všetkých rozhraniach. Možnosť špecifikovania rozhrania vyplívajúca zo zadania sa nepodarilo spolahlivo naimplementovať. Po prijatí spojenia s klientom sa odošle uvítacia správa a server prejde do fázy komunikácie s klientom.




TCP komunikácia

### 3.5.2 Komunikácia s klientom

Komunikácia s klientom prebieha v nekonečnej smyčke, kým sa klient neodpojí alebo nazašle dotaz `DONE` a prebieha v týchto krokoch:

- 1. Prijatie dotazu
- 2. Spracovania dotazu a príprava odpovede
- 3. Zaslanie odpovede a návrat na krok 1

Prijatá textová správa(dotaz) sa pre jednoduchšie spracovanie rozdelí na tokeny, obsahujúce názov dotazu a prípadné argumenty. Server následne tento tokenizovaný dotaz skontroluje, vykoná a odošle adekvátnu odpoveď klinetovy. Pre vykonanie viac príkazových dotazov (napr. `STOR` alebo `NAME`) server obsahuje stavové premenné, aby mohol tieto príkazy spolahlivo vykonať. Pre prijatie alebo odoslanie súboru nebolo potrebebné implemenotvať špeciálny protokol, kedže potrebné informácie ako meno a velkosť súboru si klient so serverom vymenia ešte pred prenosom.

## 3.6 Klient, `client.cpp/h`

Implementácia klienta je v mnohom podobná serveru, ale je omnoho kratšia nakoľko okrem odosielania a prijímania súborov a pár základných kontrol užívateľovho vstupu naobsahuje žiadnu logiku. Činnosť klienta zhrnutá v bodoch:

- 1. Pripojenie k serveru
- 2. Načítanie vstupu užívateľa
- 3. Zaslanie dotazu
- 4. Prijatie odpovede a návrat na bod 2

# 4 Návod na použitie

## 4.1 Syntax a sémantika

Hranaté zátvorky znamenajú povinný parameter a zložené zátvorky volitelný parameter.

Spustenie serveru:
`./ipk-simpleftp-server -i rozhraní -p port [-u cesta_soubor] [-f cesta_k_adresari]`
kde:

- `-i rozhraní` špecifikuje rozhranie na ktorom sa bude naslúchať (nie je implementované)
- `-p port` špecifikuje port
- `-u cesta_soubor` je absolútna cesta k súboru s databázou užívateľských účtov
- `-f cesta_k_adresari` je absolútna cesta k adresáru, kam sa budú nahrávať súbory

Spustenie klienta:
`./ipk-simpleftp-client [-h IP] -p port [-f cesta_k_adresari]]`
kde:

- `-h IP` je IPv4 či IPv6 adresa serveru
- `-p port` je port, na ktorom server naslúcha
- `-f cesta_k_adresari` je absolútna cesta k adresáru, kam sa budú sťahovať súbory

## 4.2 Prekladový systém

Súčasťou projektu je súbor makefile [3], ktorý zjednodušuje preklad a spúštanie projektu.
Návod na použitie:

- `make` Preloží projekt. Výsledok sú dva spustitelné súbory
- `make server` Preloží len server
- `make client` Preloží len klient
- `make clean` Odstráni všetky objektové a spustitelné súbory
- `make clob` Odstráni len objektové súbory

# 5 Testovanie

Pre testovanie som mal k dispozícii tri zariadenia: môj macbook, susedov macbook a referenčný systém bežiaci vo VirtualBoxe (ďalej len virtuálka). Testovanie prebiehalo na localhoste macOS, localhoste virtuálky, medzi macOS a virtuálkou a medzi dvoma macOS. Zahrnuté bolo testovanie všetkých príkazov so špeciálnym dôrazom na testovanie prenosu súborov. Testoval som postupným zadavaním príkazov, poprípade ich kombináciami a následne kontroloval, či zmeny v súborovom systéme odpovedajú odpovedi serveru.

### 5.1 Testovanie na macOS

Testovalo sa na localhoste na obidvoch verziách IP. Testovanie bolo úspešné a aplikácia bez problémov nadviazala spojenie (IPv4 aj IPv6).

### 5.2 Testovanie macOS - virtuálka

Podarilo sa úspešne nadviazať spojenie medzi klientom na virtuálke a serverom na macOS na rozhraní `en0` s IPv4. Cez IPv6 sa nepodarilo pripojiť.

### 5.3 Testovanie virtuálke

IPv4 spojenie na lovalhoste prebehlo bez problémov. Spojenie cez IPv6 loopback sa nepodarilo nadviazať.

### 5.4 Testovanie macOS - macOS

Testovanie prebiehalo medzi dvoma zariadeniami macOS. Spojenie cez IPv4 sa podarilo úspešne nadviazať. Prenos súboru bol takisto úspešný. Cez IPv6 sa spojenie nadviazať nepodarilo (ale server nebolo možné ani pingnúť cez IPv6). Tento problém sa mi pre nedostatok času nepodarilo vyriešiť.

## Referencie


[1] *File Transfer Protocol*. [online]. 1985. URL: https://www.ietf.org/rfc/rfc959.txt.

[2] *Filesystem library*. [online]. URL: https://en.cppreference.com/w/cpp/filesystem.

[3] *GNU make*. [online]. URL: https://www.gnu.org/software/make/manual/make.html.

[4] *Simple File Transfer Protocol*. [online]. 1984. URL: https://datatracker.ietf.org/doc/html/rfc913.

[5] *socket(2) — Linux manual page*. [online]. 2021. URL: https://man7.org/linux/man-pages/man2/socket.2.html.

[6] *Transmission Control Protocol*. [online]. 1981. URL: https://datatracker.ietf.org/doc/html/rfc793.