Vysoké učení technické v Brně
Fakulta informačních technologií

Počítačové komunikace a sítě

Simple File Transfer Protocol

24. apríla 2022 Alexander Okrucký (xokruc00)

Obsah

Úvod

Táto dokumentácia popisuje implementáciu, testovanie a použitie klientskej a serverovej aplikácie, ktorá slúži na prenos súborov cez Simple File Transfer Protokol. Simple FTP je definovaný v RFC 913[2].

Obe aplikácie boli naprogramované pomocou jazyka C a testované boli na operačných systémoch Fedora 35 Workstation a Ubuntu 20.04.

1 Spustenie

Pred spustením klientskej a aj serverovej aplikácie, je treba ju najprv preložiť. Na preloženie je potrebný štandardný nástroj GNU Make. Pomocou príkazu make sa obe aplikácie preložia. Preklad je uskutočnený prekladačom jazyka C, gcc.

Obe aplikácie sú CLI a nemajú grafické užívateľské prostredie. Prekladom nám vznikli dva spustitelné programy, *ipk-simpleftp-client* a *ipk-simpleftp-server*. Vyžaduje sa aby obe aplikácie boli spúšťané so správcovskými právami.

1.1 Server

Pri spustení serveru musíme zadať programu potrebné argumenty, s ktorými pracuje.
./ipk-simpleftp-server -i rozhranie -p port [**-u cesta-súbor**] [**-f cesta-k-adresary**]
Argumenty môžu byť zadané v ľubovolnom poradí a zadanie portu a rozhrania nieje povinné. Predvolené nastavenie rozhrania je *eth0* a portu *115*.

Argument **-i** označuje na akom sieťovom rozhraní bude server komunikovať. Viac rozhraní ako jedno nieje podporovaných.

Argument **-p** slúži na označenie portu, na ktorom bude server komunikovať. Číslo portu je celé desiatkové číslo v rozsahu 1-65535, vrámci týchto hodnôt.

Argument **-u** označuje cestu k databázy používateľov, ktorá je tvorená používateľskými menami a heslami vo formáte meno:heslo. Každý používateľ sa nachádza na samostatnom riadku. Pomocou tohto súboru prebieha overenie používateľa pri prihlasovaní. Ak súbor neexistuje, program skončí s chybou.

Argument **-f** označuje cestu k pracovnému adresáru v ktorom používatelia chcú primárne pracovať, avšak server nijako neobmedzuje používateľov svojvoľne sa pohybovať po adresárovom systéme serveru a vykonávať tam ľubovolnú činnosť. Ak adresár neexistuje, program skončí s chybou.

1.2 Klient

Pri spustení serveru musíme zadať programu potrebné argumenty, s ktorými pracuje.
./ipk-simpleftp-client [**-h IP**] **-p port** [**-f cesta-k-adresáru**]
Argument port je voliteľný a jeho predvolené nastavenie je 115. Argumenty môžu byť zadané v ľubovolnom poradí.

Argument **-h** očakáva IPv4 alebo IPv6 adresu serveru. Viac IP adries nieje podporovaných.

Argument **-p** slúži na označenie portu, na ktorom server komunikuje. Číslo portu je celé desiatkové číslo v rozsahu 1-65535, vrámci týchto hodnôt.

Argument **-f** označuje cestu k pracovnému adresáru na systéme klienta. Do tohto adresáru sa budú sťahovať súbory zo serveru. Tento adresár nieje možné počas behu programu zmeniť, preto je odporúčané všetky súbory, ktoré chcete nahrať na server, vložiť do tohto pracovného adresára. Ak adresár neexistuje, program skončí s chybou.

2 Funkcionalita

Server a klient medzi sebou dokážu posielať súbory a klient môže sa pohybovať v adresárovom systéme serveru. Taktiež môže klient mazať súbory a adresáre. Ukončenie serveru sa dá klávesovou skratkou *CTRL+C* v termináli, v ktorom je spustený server. Klienta je taktiež možné ukončiť pomocou klávesovej skratky *CTRL+C* v termináli, v ktorom je klient spustený. Ukončenie klienta je taktiež možné pomocou Simple FTP príkazu *DONE*.

3 Implementácia

Implementácia spočívala z vytvorenia dvoch programov, klienta a servera. Server počúva na danom rozhraní a porte a čaká na prichádzajúce pripojenie. Skoro všetky príkazy spracováva server samotný, klient si len skontroluje či zadaný príkaz je validný, argumenty príkazov klient nekontroluje, to má na starosti server.

Ak server odhalí chybu tak odošle klientovi odpoveď začínajúcu znakom '-' a so správou popisujúcou problém, ktorý nastal. Ak sa operácia vykoná úspešne tak server odošle odpoveď začínajúcu na '+' a správou popisujúcou čo sa podarilo vykonať. Správa začínajúca znakom '!' znamená, že pri prihlasovaní bol prvý zadaný prihlasovací údaj správny a očakáva druhý. Pri prihlasovaní je možné najprv odoslať heslo a až potom heslo. Príkazy a ich argumenty sú citlivé na veľkosť písmen.

3.1 Vytvorenie spojenia

Po úspešnom spustení serveru je možné sa naň pripojiť prostredníctvom klienta. Nadviazané spojenie prebieha cez TCP. To znamená, že po celú dobu komunikácie klient - server ostáva pripojenie aktívne, až na konci, po vypnutí zo strany klienta alebo servera je prerušené.

Po úspešnom pripojení klienta pozdraví hláška '+welcome, SFTP service active'. Po nej môže klient písať príkazy do terminálu a tie budú následne odoslané na server, kde budú spracované a vyhodnotené.

3.2 Prihlásenie

Bez úspešného prihlásenia klient nemôže pracovať so súbormi. Pre pomoc s prihlasovaním na server je v klientovi implementovaný príkaz *LET-ME-IN*, tento príkaz je spracovaný na klientovi a neodošle sa serveru.

Prihlasovanie nemá striktne dané poradie prihlasovacích údajov. To znamená že klient môže aj ako prvý údaj pri prihlasovaní zadať heslo pomocou príkazu *PASS*. Odoslanie prihlasovacieho mena môže uskutočniť klient pomocou príkazu *USER* alebo *ACCT*. Oba príkazy sú implementované identicky a sú si ekvivalentné.

PASS *heslo*
USER *meno*
ACCT *meno*

3.3 Práca so súbormi a adresármi

Klient dokáže prechádzať súborovým systémom serveru pomocou príkazu *CDIR*, implementácia tohto príazu je zhodná s Unixovým príkazom *cd*. Pri zadaní *CDIR ..* klient prejde do nadradeného adresáru.

Pomocou príkazu *LIST* môže klient vypísať obsah adresára. *LIST* má povinný argument *F* alebo *V*, rozdiel medzi týmito variantami príkazu sú v štruktúrovaní výstupu.

Príkaz *KILL* zmaže zadaný súbor alebo adresár. Tento príkaz je ekvivalentný s príkazom z Unixu *rm -r*. Povinný argument je súbor alebo adresár, poprípade cesta k nemu. Príkaz *KILL* podporuje rekurzívne mazanie.

Zmena názvu súboru alebo adresáru je možná pomocou príkazu *NAME* a *TOBE*. Príkaz *NAME* má argument starý názov súboru, adresáru alebo cestu k nemu. Po úspešnom vykonaní príkazu *NAME* je možné použiť príkaz *TOBE*, ktorý má argument nový názov súboru. Príkaz *TOBE* daný súbor alebo adresár premenuje pomocou Unixového príkazu *mv*.

Samotný protokol *SFTP* nepodporuje príkaz *HOME* ale tento príkaz je na servery podporovaný. Príkaz *HOME* presunie klienta naspať do základného, zadaného, pracovného adresára na servery.

Po prihlásení môže klient ukončiť spojenie so serverom pomocou príkazu *DONE*.

CDIR *adresár*
LIST *F* | *V*
KILL *súbor* | *adresár*
NAME *súbor* | *adresár*
TOBE *názov*
HOME
DONE

3.4 Prenos súborov

V pôvodnej implementácii som počítal s implementáciou príkazu *TYPE* tak ako je zadané v špecifikácii, a to aby sa dalo meniť typ reprezentácie dát, ktoré budú odosielané zo serveru pri prenose súborov. Avšak nepodarilo sa mi rozumne implementovať *NETASCII* a preto príkaz *TYPE* je prebytočný avšak podporovaný. Príkaz TYPE nemá funkcionalitu.

Príkaz *RETR* označí súbor, ktorý klient chce prijať zo serveru. Server odošle klientovi veľkosť daného súboru v bajtoch. Ak sa používateľ rozhodne že daný súbor chce prijať tak musí použiť príkaz *SEND*. Ak by sa klient rozhodol že ho nechce, tak použije príkaz *STOP*. Daný súbor sa prenesie a uloží do pracovného adresára u klienta.

Prenos súboru na server je realizovaný príkazom *STOR*, ktorý má povinné dva argumenty. Prvý argument špecifikuje ako sa má zapísať prenesený súbor. Argument *NEW* a *APP* sú ekvivalentné, ak daný súbor už v adresári na servery existuje tak k nemu obsah nového súboru pripíše na koniec súboru. Ak daný súbor neexistuje tak bude vytvorený nový. Argument *OLD* prepíše starý súbor ak taký súbor už existuje, ak neexistuje tak vytvorí nový. Po príkaze *STOR* by mal nasledovať príkaz *SIZE*, ktorý odošle serveru veľkosť súboru v bajtoch. Prenos súboru nastane až po príkaze *SIZE*.

TYPE *A* | *B* | *C*
RETR *súbor*
SEND
STOP
STOR *NEW* | *OLD* | *APP súbor*
SIZE *veľkosť súboru*

4 Implementačné detaily

V tejto sekcii je popísaný spôsob implementácie vytvorenia pripojenia, odosielanie a spracovávanie príkazov v programovacom jazyku C.

4.1 Vytvorenie pripojenia

Pre vytvorenie pripojenia najprv treba správne vytvoriť a nakonfigurovať socket[3]. Za týmto účelom som využil knižnicu *sys/socket.h*. Socket som nakonfiguroval aby mohol byť port a aj adresa znovu využiteľné. Socket sa pripojí na zadané rozhranie a pripojí sa na špecifikovaný port. Server podporuje IPv4 aj IPv6 ako dual-stack, to znamená že IPv4 komunikáciu namapuje do IPv6 packetu. To znamená na sockete musela byť vypnutá možnosť IPV6_V6ONLY. Tento socket je TCP, to znamená že vie prijať iba TCP pripojenie.

Po úspešnom vytvorení a pripojení serveru na daný port na danom rozhraní je možné sa na server pripojiť.

Implementácia pripojenia klienta je oproti serveru jednoduchšia. Klient dostane cieľovú IP adresu, podporuje aj IPv4 alebo IPv6. Zo zadanej IP adresy a portu si pomocou funkcie *getaddrinfo(...)*[4] zistí dodatočné parametre serveru, z ktorých si vytvorí socket a pomocou neho sa pripojí na server.

4.2 Server - klient komunikácia

Vzhľadom na to že komunikácia prebieha cez TCP tak musí byť udržiavaná po celú dobu, dokým ju používateľ nevypne. Klient napíše príkaz do terminálu na *stdin* a ten sa overí na klientovi a následne odošle na server, pomocou príkazu *send()* Server správu načíta cez funkciu *recv()* a spracuje príkaz. Overí či adresár alebo súbor existujú.

5 Testovanie

Dôležitá časť projektu je aj testovanie programu a komunikácie medzi serverom a klientom. Testovanie prebiehalo cez virtualizovanú sieť ale aj cez skutočnú lokálnu sieť. Packety som si kontroloval pomocou nástroja Wireshark.

5.1 Vitualizovaná sieť

Virtualizovanú sieť som si vytvoril v programe Virtualbox. Ako typ siete som si vybral klientom hosťovanú[1]. Na sieti boli dva virtualizované identické operačné systémy a to Ubuntu 20.04. Jeden slúžil ako server a druhý ako klient. Prvý test testoval pripojenie cez IPv4.

Test IPv6 pripojenia prebiahal pomocou súkromnej IPv6 adresy za ktorou musí nasledovať % a rozhranie serveru.

Ďalší test spočíval v testovaní ostatných rozhraní na servery, či náhodou server není pripojený aj na ostatné rozhrania. Vytvoril som si ďalší virtuálny sieťový adaptér a priradil ho k operačnému systému na ktorom bežal server.

Taktiež bolo testované na Ubuntu 20.04 preloženie projektu vzhľadom na to že Fedora 35 beží na najnovších technológiách a niektoré funkcie fungujú odlišne.

5.2 Lokálna sieť

Ďalším významným testovaním bolo na vlastnej domácej lokálnej siete. Naša sieť podporuje aj komunikáciu cez IPv6. Test prebiehal na stolnom počítači so systémom Fedora 35 Workstation, ktorý slúžil ako server. Pripojený bol cez ethernet port. Ako klient slúžil notebook s identickým systémom Fedora 35 Workstation. Notebook bol do siete pripojený cez ethernet port a nie cez Wi-Fi, lebo na našej sieti

sú zariadenia pripojené cez Wi-Fi na inej VLAN ako zariadenia pripojené cez ethernet. Toto riešenie je z dôvodu bezpečnosti aby sa návšteva oddelila od našich zariadení. Medzi serverom a klientom bol pripojený router.

Test IPv6 prebiehal cez súkromnú IPv6 adresu ale aj cez verejnú IPv6 adresu daného rozhrania.

Literatúra

[1] DePaulUniversity: Host-only Networking in VirtualBox. [online], [vid. 2022-4-17].
URL https://condor.depaul.edu/glancast/443class/docs/vbox$_h ost - only_s etup.html$

[2] LOTTOR, M. K.: Simple File Transfer Protocol. [online], September 1984, [vid. 2022-4-17].
URL https://datatracker.ietf.org/doc/html/rfc913

[3] man7.org: socket(2) — Linux manual page. [online], 2021-03-22, [vid. 2022-4-17].
URL https://man7.org/linux/man-pages/man2/socket.2.html

[4] man7.org: getaddrinfo(3) — Linux manual page. [online], 2021-08-27, [vid. 2022-4-17].
URL https://man7.org/linux/man-pages/man3/getaddrinfo.3.html