VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

TFTP CLIENT AND SERVER
THESIS TITLE

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE — **DENIS NOVOSÁD**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN DOLEJŠKA**
SUPERVISOR

BRNO 2023

Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

Abstract

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

Keywords

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Citace

NOVOSÁD, Denis. *TFTP client and server*. Brno, 2023. Semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Dolejška

TFTP client and server

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Denis Novosád
14. listopadu 2023

Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

Obsah

Seznam obrázků

Kapitola 1

Úvod

TFTP je jednoduchý, textový síťový protokol určený pro přenos souborů mezi zařízeními v počítačových sítích. Byl vyvinut tak, aby byl co nejjednodušší a mohl být implementován na různých platformách s minimálním úsilím.

TFTP vyniká svou jednoduchostí a nízkou náročností na zdrojové prostředky, což ho činí vhodným pro situace, kde je potřeba rychlý a spolehlivý přenos dat, ale složitost a zabezpečení jako u jiných protokolů, například FTP (File Transfer Protocol), nejsou potřebné. To znamená, že TFTP je často používán v různých prostředích, jako jsou embedded systémy, zařízení s omezenými zdroji nebo v jednoduchých síťových scénářích, kde jednoduchost a nízká latence mají prioritu.

Navzdory své jednoduchosti TFTP stále slouží jako základní nástroj pro přenos souborů v mnoha síťových aplikacích a je důležitým prvkem v oblasti správy souborů a aktualizací firmwaru v zařízeních IoT (Internet of Things) a v síťovém bootování. Z tohoto důvodu má porozumění TFTP a jeho implementace zásadní význam pro vývojáře a síťové inženýry, kteří pracují s tímto protokolem. V našem projektu se zaměřujeme na implementaci TFTP klienta a serveru, které umožní přenos souborů mezi těmito dvěma entitami v prostředí počítačové sítě.

Kapitola 2

Abstrakt

Tento projekt představuje implementaci klienta a serveru pro protokol TFTP (Trivial File Transfer Protocol). TFTP je jednoduchý síťový protokol sloužící k přenosu souborů mezi počítači v lokální nebo vzdálené síti. Klient a server byly vyvinuty v programovacím jazyce C++ a navrženy tak, aby byly snadno použitelné a mohly být integrovány do existujících síťových aplikací.

Klient umožňuje uživatelům stahovat soubory ze vzdáleného serveru a nahrávat soubory na server. Podporuje základní operace čtení (RRQ - Read Request) a zápisu (WRQ - Write Request) souborů, a to včetně volitelných parametrů, jako je velikost bloku (blksize), velikost přenášeného souboru (tsize) a interval znovuodesílání packetů (timeout). Klient byl navržen tak, aby byl konfigurovatelný a umožňoval uživatelům specifikovat různé parametry pro přenos souborů.

Server na druhé straně přijímá požadavky od klientů a provádí operace čtení a zápisu souborů.

Tato implementace TFTP klienta a serveru poskytuje nástroje pro snadný přenos souborů v rámci sítě a může být využita v různých síťových aplikacích. Jejich použití je dokumentováno v následujících sekcích této technické zprávy.

Tento projekt byl vytvořen s ohledem na jednoduchost, efektivitu a spolehlivost, což jsou klíčové vlastnosti pro TFTP klienta a server.

Kapitola 3

Problematika návrhu aplikace

Kód klienta TFTP obsahuje několik funkcí a logiku pro odesílání a přijímání souborů.

3.1 Základní vlastnosti TFTP

- UDP komunikace: TFTP využívá UDP pro komunikaci, což znamená, že neposkytuje zajištění doručení dat ani zajištění integrity dat. Reliabilitu musí zajistit samotná aplikace na vyšší vrstvě.
- Operace: TFTP podporuje dvě základní operace - čtení (RRQ - Read Request) a zápis (WRQ - Write Request). Klient může požádat server o čtení souboru nebo o zápis do souboru.
- Datové pakety: Data jsou přenášena v datových paketech. Každý datový paket obsahuje blok dat a číslo bloku, což umožňuje klientovi a serveru sledovat postup přenosu.
- ACK (Acknowledgment) pakety: Po přijetí datového paketu server nebo klient odpoví potvrzením (ACK) s číslem potvrzeného bloku.
- OACK (Option Acknowledgment) pakety: OACK mohou být použity při zahájení komunikace k dohodnutí různých parametrů, například velikosti bloku.

Kapitola 4

Návrh serveru TFTP

4.1 Funkce serveru:

- Čekání na klienta: Server bude naslouchat na určeném portu na příchozí TFTP požadavky.
- Příjem RRQ a WRQ: Po obdržení RRQ (Read Request) nebo WRQ (Write Request) od klienta bude server připraven ke čtení nebo zápisu souboru.
- Čtení souboru (RRQ): Pokud obdrží RRQ, server otevře požadovaný soubor, přečte data a odešle je klientovi v datových paketech. Server čeká na potvrzení (ACK) od klienta a pokračuje v posílání dalších bloků dat, dokud není celý soubor přenesen.
- Zápis do souboru (WRQ): Pokud obdrží WRQ, server přijme data od klienta a zapíše je do požadovaného souboru. Server odešle potvrzující ACK po přijetí každého datového bloku a čeká na další data od klienta.
- Ovládání chyb: Server bude schopen ovládat chyby, například v případě, že soubor neexistuje nebo že dojde k timeoutu při čekání na ACK od klienta.

Kapitola 5

Návrh klienta TFTP

5.1 Funkce klienta:

- Odesílání RRQ a WRQ: Klient bude schopen odeslat RRQ nebo WRQ na server a začít přenášet soubory.
- Čekání na DATA a ACK: Po odeslání RRQ nebo WRQ klient čeká na příchozí datové a potvrzující (ACK) pakety od serveru.
- Čtení souboru (RRQ): Pokud klient obdrží DATA paket, zapíše data do lokálního souboru a odešle ACK s potvrzením.
- Zápis do souboru (WRQ): Pokud klient obdrží ACK, odešle další datový paket. Tento proces se opakuje, dokud není celý soubor odeslán.
- Ovládání chyb: Klient bude schopen detekovat různé chyby a provádět akce, jako je opětovné odesílání dat nebo ukončení přenosu.

Kapitola 6

Struktura datových paketů:

- Přenos souborů: Klient musí být schopen přenášet soubory ze serveru na lokální stroj (čtení) a z lokálního stroje na server RRQ (Read Request): opcode (2 bajty) + soubor (n) + 0 (1 bajt) + režim (n) + 0 (1 bajt) + volby (n) + 0 (1 bajt). Tento paket požaduje čtení souboru.
- WRQ (Write Request): opcode (2 bajty) + soubor (n) + 0 (1 bajt) + režim (n) + 0 (1 bajt) + volby (n) + 0 (1 bajt). Tento paket požaduje zápis do souboru.
- DATA: opcode (2 bajty) + číslo bloku (2 bajty) + data (n). Tento paket obsahuje data k přenosu.
- ACK: opcode (2 bajty) + číslo potvrzeného bloku (2 bajty). Tento paket potvrzuje přijetí datového bloku.
- ERROR: opcode (2 bajty) + kód chyby (2 bajty) + zpráva o chybě (n) + 0 (1 bajt). Tento paket obsahuje informace o chybě a jejím kódu.

Kapitola 7

Popis options

blksize

Parametr **blksize** umožňuje nastavit velikost datových bloků, které klient požaduje nebo přijímá během přenosu souboru. Tento parametr může být použit k optimalizaci rychlosti přenosu a minimalizaci režie spojenou s počtem přenosů. Implementace klienta umožňuje nastavit velikost datových bloků v rozmezí od 8 do 65464 bytů.

tsize

Parametr **tsize** je používán k určení celkové velikosti přenášeného souboru. Klient může použít tento parametr pro monitorování průběhu přenosu a zobrazování procenta přenesených dat. Pokud je tento parametr použit, klient bude pravidelně aktualizovat a zobrazovat procentuální ukazatel průběhu přenosu na standardním výstupu.

timeout

Parametr **timeout** určuje maximální dobu čekání na odpověď od serveru v sekundách. Pokud server neposkytne odpověď včas, klient provede opakované pokusy o znovu odeslání požadavku.

Kapitola 8

Implementace Serveru

8.0.1 Inicializace a nastavení socketu

Program začíná vstupní funkcí `main`, kde se zpracovávají příkazové řádky a získávají se parametry, jako je port a kořenový adresář pro TFTP server. Následně se vytváří socket (UDP) pomocí funkce `socket()`, který bude použit pro komunikaci s klienty.

8.0.2 Příjem TFTP požadavků od klientů

Server čeká na příchozí TFTP požadavky od klientů pomocí funkce `recvfrom()`. Požadavky jsou přijímány ve formě TFTP paketů, které obsahují opcode (kód operace), název souboru, režim přenosu a volitelné parametry.

8.0.3 Rozpoznání TFTP operace

Server rozpoznává typ operace (čtení nebo zápis) na základě opcode ve zprávě. Implementuje operace RRQ (čtení) a WRQ (zápis).

8.0.4 Obsluha operací čtení a zápisu

Pro operaci čtení (RRQ) server zpracovává požadavek klienta na čtení souboru. Pro operaci zápisu (WRQ) server zpracovává požadavek klienta na zápis souboru.

8.0.5 Zpracování volitelných parametrů

Pokud jsou přítomny volitelné parametry v požadavku klienta (např. velikost bloku, timeout, velikost přenosu), server tyto parametry zpracuje a použije je při přenosu souboru. Parametry jsou přijímány ve formátu "název=hodnota".

8.0.6 Zajištění integrity přenosu

Server zajišťuje integritu přenosu dat tím, že pošle potvrzení (ACK) klientovi poté, co obdrží každý blok dat. Server také očekává potvrzení od klienta po odeslání datového bloku.

8.1 Příjem souboru (Operace RRQ - čtení souboru):

1. **Klient odesílá RRQ packet:**

- Klient začíná tím, že odešle RRQ (Read Request) packet serveru. Tento packet obsahuje opcode 1 pro čtení, název požadovaného souboru a režim přenosu (např. "octet"pro binární přenos).

2. **Server přijímá RRQ packet:**
   - Server čeká na příchozí packet pomocí funkce `recvfrom()`. Přijatý packet je zpracován a opcode je identifikován jako RRQ.
3. **Kontrola oprávnění a existence souboru:**
   - Server zkontroluje, zda klient má oprávnění k čtení tohoto souboru. Může to zahrnovat kontroly přístupových práv nebo seznam povolených souborů.
   - Server také zkontroluje, zda soubor existuje. Pokud ne, server pošle odpovídající packet s chybovým kódem zpět klientovi.
4. **Čtení souboru a odesílání dat:**
   - Pokud je vše v pořádku, server otevře požadovaný soubor pro čtení.
   - Data ze souboru jsou čtena v blocích s určitou velikostí (diktovanou klientem nebo výchozí nastavenou). Data jsou čtena do bufferu.
5. **Vytvoření a odeslání DATA packetu:**
   - Server vytvoří DATA packet s obsahem z bufferu a zvýšeným číslem bloku.
   - Tento packet je odeslán klientovi pomocí funkce `sendto()`.
6. **Čekání na potvrzení (ACK) od klienta:**
   - Po odeslání DATA packetu server čeká na potvrzení (ACK) od klienta. Potvrzení obsahuje číslo bloku, který byl právě úspěšně přijat.
   - Pokud nedojde k potvrzení v určitém časovém limitu nebo dojde k chybě, server může znovu odeslat DATA packet nebo ukončit přenos.
7. **Opakování kroků 4-6 až do dokončení přenosu:**
   - Kroky 4-6 jsou opakovány, dokud není celý soubor úspěšně přenesen.
8. **Uzavření souboru a ukončení přenosu:**
   - Po úspěšném přenesení celého souboru server uzavře soubor, ukončí přenos a čeká na další požadavky od klientů.

8.2 Odesílání souboru (Operace WRQ - zápis souboru):

1. **Klient odesílá WRQ packet:**
   - Klient začíná tím, že odešle WRQ (Write Request) packet serveru. Tento packet obsahuje opcode 2 pro zápis, název cílového souboru a režim přenosu (např. "octet"pro binární přenos).
2. **Server přijímá WRQ packet:**

- Server čeká na příchozí packet pomocí funkce `recvfrom()`. Přijatý packet je zpracován a opcode je identifikován jako WRQ.

3. **Kontrola oprávnění a existence souboru:**
   - Server zkontroluje, zda klient má oprávnění k zápisu do tohoto souboru. To může zahrnovat kontroly přístupových práv nebo seznam povolených souborů.
   - Server také zkontroluje, zda soubor již existuje. Pokud ano, server pošle odpovídající packet s chybovým kódem zpět klientovi.
4. **Otevření souboru pro zápis:**
   - Pokud je vše v pořádku, server vytvoří nebo otevře cílový soubor pro zápis.
5. **Přijímání a ukládání DATA packetů:**
   - Server přijímá DATA packety od klienta. Každý packet obsahuje určité číslo bloku a data, která mají být zapsána do souboru.
   - Data z packetu jsou zapsána do souboru na serveru.
6. **Vytvoření a odeslání ACK packetu:**
   - Po úspěšném zápisu dat do souboru server vytvoří ACK (Acknowledgment) packet s číslem

8.2.1 Ošetření výjimek a chyb

Server má mechanismy pro zacházení s chybami a výjimkami, jako je neexistence souboru, nedostatek místa na disku, nedostatek paměti, časový limit pro přenos dat atd.

8.2.2 Ukončení serveru

Server lze ukončit pomocí signálu SIGINT (Ctrl+C). Při zachycení tohoto signálu je provedeno ukončení serveru a je provedena čistá terminace.

Kapitola 9

Implementace klienta

9.1 Celkový Průběh Programu

9.1.1 Zpracování Příkazové Řádky

Klient začíná zpracováním příkazové řádky, kde uživatel zadává parametry pro komunikaci s TFTP serverem. Parametry mohou zahrnovat:

- `-h hostname`: Hostname (adresa) TFTP serveru.
- `-p port`: Port, na kterém server naslouchá (výchozí hodnota je 69).
- `-f filepath`: Cesta k souboru na serveru, který má být stažen (pouze pro příjem).
- `-t dest_filepath`: Cesta k lokálnímu souboru, kam má být uložen stažený soubor (pouze pro příjem).
- `-option`: Volitelný parametr pro přijetí OACK (Option Acknowledgment) od serveru, který obsahuje další informace o přenosu.

9.1.2 Inicializace Datových Struktur

Na začátku programu jsou inicializovány datové struktury pro správu různých parametrů a stavů komunikace.

- `TFTPOparams`: Struktura pro ukládání parametrů pro TFTP komunikaci, jako je velikost bloku (blksize), timeout a celková velikost (tsize).
- Další proměnné pro ukládání cest k souborům, režim komunikace (např. "octet") a volitelné parametry (přes `-option`).

9.1.3 Vytvoření Socketu

Klient vytváří UDP socket pro komunikaci se serverem. Socket je inicializován funkcí `socket()`.

Pokud vytvoření socketu selže, program ukončí s chybou.

9.2 Odesílání souboru

(a) **Odeslání Write Request (WRQ):**

- Klient začíná odesláním Write Request (WRQ) na server. WRQ obsahuje název cílového souboru na serveru a režim komunikace (např. "octet").

(b) **Čtení lokálního souboru a odesílání datových paketů:**

- Po obdržení potvrzení (ACK) od serveru začne klient číst data ze svého lokálního souboru, který má být odeslán.
- Data jsou čtena v blocích, kde velikost bloku je určena parametrem `blksize`.
- Data jsou pak uložena do datového paketu (DATA packet) a odeslána na server.
- Klient čeká na potvrzení (ACK) od serveru pro každý odeslaný datový blok.

(c) **Potvrzení (ACK) od serveru:**

- Po odeslání každého datového paketu klient čeká na potvrzení (ACK) od serveru.
- Potvrzení obsahuje číslo bloku, který byl úspěšně přijat na straně serveru.
- Pokud klient nepřijme potvrzení včas nebo obdrží chybové potvrzení, může to vést k opakovanému odeslání stejného datového paketu.

(d) **Kontrola bloků a konec přenosu:**

- Klient udržuje číslo bloku, který má být odeslán v následujícím datovém paketu.
- Když je všechna data odeslána a potvrzena, přenos souboru končí.

9.3 Přijímání souboru

(a) **Odeslání Read Request (RRQ):**

- Klient začíná odesíláním Read Request (RRQ) na server. RRQ obsahuje název cílového souboru na serveru a režim komunikace (např. "octet").

(b) **Přijímání datových paketů a odesílání potvrzení (ACK):**

- Po odeslání RRQ klient začne přijímat datové pakety od serveru.
- Klient čeká na příjem datového paketu a následně odesílá potvrzení (ACK) s číslem bloku, které obsahovalo přijatý datový paket.
- Potvrzení zajistí, že server ví, který datový blok byl úspěšně přijat.

(c) **Zápis přijatých dat do lokálního souboru:**

- Přijatá data z datových paketů jsou zapisována do lokálního souboru na straně klienta.
- Data jsou zapisována do souboru v pořadí, jak přicházejí od serveru.

(d) **Kontrola bloků a konec přenosu:**

- Klient udržuje číslo očekávaného bloku, který má být přijat od serveru.
- Když přijme datový paket s číslem bloku, který očekává, odesílá potvrzení (ACK) a zapisuje data do lokálního souboru.

- Přenos souboru končí, když datový paket má velikost menší než velikost bloku (tj. je přenášen poslední blok souboru).

(e) **Opakování potvrzení (ACK):**

- Pokud klient nepřijme datový paket včas nebo obdrží chybový datový paket, může to vést k opakovanému odeslání potvrzení (ACK) se stejným číslem bloku.
- Opakované potvrzení pomáhá zajistit, že server znovu nepošle stejný datový blok.

9.4 Odesílání souboru

(a) **Odeslání Write Request (WRQ):**

- Klient začíná odesláním Write Request (WRQ) na server. WRQ obsahuje název cílového souboru na serveru a režim komunikace (např. "octet").

(b) **Čtení lokálního souboru a odesílání datových paketů:**

- Po obdržení potvrzení (ACK) od serveru začne klient číst data ze svého lokálního souboru, který má být odeslán.
- Data jsou čtena v blocích, kde velikost bloku je určena parametrem `blksize`.
- Data jsou pak uložena do datového paketu (DATA packet) a odeslána na server.
- Klient čeká na potvrzení (ACK) od serveru pro každý odeslaný datový blok.

(c) **Potvrzení (ACK) od serveru:**

- Po odeslání každého datového paketu klient čeká na potvrzení (ACK) od serveru.
- Potvrzení obsahuje číslo bloku, který byl úspěšně přijat na straně serveru.
- Pokud klient nepřijme potvrzení včas nebo obdrží chybové potvrzení, může to vést k opakovanému odeslání stejného datového paketu.

(d) **Kontrola bloků a konec přenosu:**

- Klient udržuje číslo bloku, který má být odeslán v následujícím datovém paketu.
- Když je všechna data odeslána a potvrzena, přenos souboru končí.

(e) **Opakování odeslání v případě chyby:**

- Pokud klient nepřijme potvrzení (ACK) v určeném časovém limitu, opakuje odeslání stejného datového paketu.
- Toto opakování pokračuje, dokud klient nepřijme potvrzení nebo nedosáhne maximálního počtu pokusů.
- Opakované odesílání pomáhá zajistit spolehlivý přenos dat.

9.4.1 Komunikace s Serverem

Klient vstupuje do smyčky, kde komunikuje se serverem. V případě odesílání souboru klient načítá data z lokálního souboru a posílá je na server ve formě datových paketů

(DATA packets). Po odeslání každého datového paketu čeká na potvrzení od serveru (ACK packet).

V případě přijímání souboru klient přijímá data od serveru ve formě datových paketů. Po přijetí každého datového paketu odesílá potvrzení (ACK packet) zpět na server. Pokud jsou použity volitelné parametry (OACK), klient je posílá s datovými pakety a očekává ACK potvrzení.

9.4.2 Ukládání Přijatých Dat

Přijatá data (datové pakety) jsou ukládána do lokálního souboru (při přijetí) nebo odesílána na server (při odesílání).

9.4.3 Konec Přenosu a Uzavření Socketu

Přenos souboru končí, když jsou všechny bloky přeneseny nebo když dojde k chybě. Po dokončení přenosu klient uzavře socket a v případě přijetí souboru také uzavře výstupní soubor.

9.5 Datové Struktury

9.5.1 TFTPOparams

Struktura `TFTPOparams` slouží k ukládání parametrů pro TFTP komunikaci. Obsahuje následující položky:

- `blksize`: Velikost bloku (block size) pro přenos dat. Toto číslo ovlivňuje, kolik datových bytů je přeneseno najednou v jednom datovém paketu.
- `timeout_max`: Maximální doba čekání na odpověď od serveru (timeout) ve vteřinách. Pokud server neodpoví do této doby, klient opakuje pokusy nebo ukončí přenos.
- `transfersize`: Celková velikost (transfersize) souboru, pokud je tato informace k dispozici. Slouží k monitorování pokroku přenosu.

9.5.2 Další Proměnné

Kromě `TFTPOparams` klient také používá další proměnné pro ukládání cest k souborům, režimu komunikace (např. "octet") a volitelných parametrů předaných přes `-option`.

Kapitola 10

Základní informace

10.1 Program 1: TFTP Server

TFTP (Trivial File Transfer Protocol) server je síťový protokol pro jednoduchý přenos souborů mezi klienty a serverem. Zde jsou základní informace o TFTP serveru:

- **Název:** TFTP Server
- **Účel:** Poskytuje možnost klientům stahovat a odesílat soubory pomocí TFTP.
- **Programovací Jazyk:** C++
- **Hlavní Funkce:** Obsluhuje žádosti klientů o stahování a odesílání souborů, včetně odesílání bloků dat a přijímání potvrzení.

10.2 Program 2: TFTP Client

TFTP klient je síťový protokol pro jednoduchý přenos souborů mezi klienty a serverem. Zde jsou základní informace o TFTP klientovi:

- **Název:** TFTP Client
- **Účel:** Poskytuje možnost klientům stahovat a odesílat soubory pomocí TFTP.
- **Programovací Jazyk:** C++
- **Hlavní Funkce:** Zpracování příkazového řádku, odesílání žádostí o stahování a odesílání souborů na server, přijímání dat a potvrzení.

Kapitola 11

Návod k použití

11.1 Kompilace

Před použitím klienta a serveru je nutné provést jejich kompilaci. Pro kompilaci použijte přiložený Makefile. Otevřete terminál a přejděte do hlavního adresáře. V něm spusťte následující příkaz:

```
make
```

Tímto způsobem by měly být v adresáři bin vytvořeny spustitelné soubory pro klienta (`tftp-client`) a server (`tftp-server`).

11.2 Spuštění klienta

Pro spuštění TFTP klienta použijte následující příkaz:

```
tftp-client -h hostname [-p port] [-f filepath] -t dest_filepath
[--option "parametr hodnota"]
```

Kde:

- `hostname` je IP adresa nebo doménový název vzdáleného serveru.
- `-p port` (volitelný) je port vzdáleného serveru. Pokud není specifikován, použije se výchozí hodnota podle specifikace.
- `-f filepath` (volitelný) je cesta ke stahovanému souboru na serveru (download). Pokud není specifikován, klient použije obsah stdin (upload).
- `-t dest_filepath` je cesta, pod kterou bude soubor na vzdáleném serveru nebo lokálně uložen.
- `-option "parametr hodnota"` (volitelný) značí, že můžete specifikovat volitelné parametry, které jsou implementovány v klientovi. Například `-option "blksize 1024"` nastaví velikost bloku na 1024 bajtů.

11.3 Spuštění serveru

Pro spuštění TFTP serveru použijte následující příkaz:

```
tftp-server [-p port] root_dirpath
```

Kde:

- `-p port` (volitelný) je místní port, na kterém bude server očekávat příchozí spojení. Pokud není specifikován, použije se výchozí hodnota.
- `root_dirpath` je cesta k adresáři, pod kterým se budou ukládat příchozí soubory na serveru.

Literatura

[1] Sollins, K. R., & W. L. (1992). *TFTP Protocol (Revision 2)*. RFC 1350.

[2] Malkin, G., & Harkin, A. (1998). *TFTP Option Extension*. RFC 2347.

[3] Malkin, G. (1998). *TFTP Blocksize Option*. RFC 2348.

[4] Malkin, G. (1998). *TFTP Timeout Interval and Transfer Size Options*. RFC 2349.