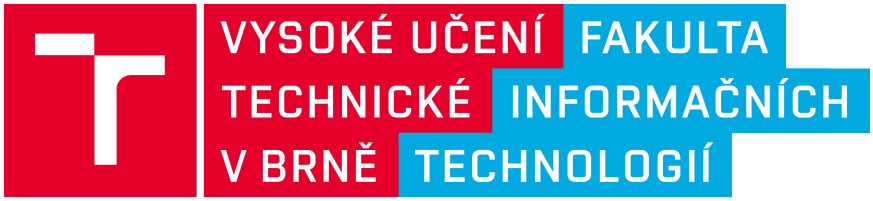


Technická zpráva

TFTP – ISA 2023/24

Veronika Nevařilová (xnevar00)

20. listopadu 2023

Obsah

1 Úvod **2**

2 Návrh aplikace **2**

3 Implementace **3**
3.1 Klientská část 3
3.2 Serverová část 4

4 Testování **4**

5 Základní informace o programu **6**

6 Návod na použití **6**
6.1 Spouštění klienta 6
6.2 Spouštění serveru 6

7 Literatura **6**

1 Úvod

TFTP je jednoduchý protokol pro přenos souborů. Přenos probíhá pomocí UDP protokolu. Vzhledem k tomu, že při přenosu pomocí UDP není zajištěna garance doručení packetů[1], je možné, že některé packety nebudou doručeny a odesílatel to nijak nezjistí. Proto TFTP svou implementací nahrazuje toto potvrzení o přijetí, a tím zabezpečuje, že není možné přenést soubor pro obě strany úspěšně, když byl nějaký packet při přenosu ztracen.

Přenos iniciuje klient, který zašle na port serveru Read nebo Write Request (pokud port není explicitně nastaven, za výchozí hodnotu, na které server poslouchá, se používá hodnota 69). Server obdržení této zprávy potvrdí tak, že buď pošle Acknowledgement packet (při Write Requestu) nebo přímo Data packet (při Read Requestu). Každý datový packet má své číslo. Toto číslo použije příjemce dat v odesílaném Acknowledgement packetu, čímž potvrdí, že daný blok dat obdržel. Přenos končí tehdy, když příjemce obdrží blok s velikostí menší než velikost bloku (výchozí hodnota 512 bytů). Pokud by se náhodou konec souboru zarovnal s velikostí bloku, bude jako poslední packet zaslán prázdný datový packet.

Typy packetů používaných při přenosu souboru pomocí TFTP:

- RRQ/RRW – pro zahájení komunikace, odesílá vždy klient
- DATA – obsahují samotná data přenášeného souboru
- ACK – slouží k potvrzení přijatých dat
- ERROR – zasílán, pokud nastala chyba

Při implemetnaci TFTP lze také podporovat několik rozšíření. Rozšíření podporované serverem v tomto projektu jsou:

- Option Extension[2] – Podporuje zasílání klientem nastavených voleb (Option) v Read Request a Write Request packetu.
- Blocksize Option[3] – Povoluje volbu velikosti bloku pro data a tím možnost například rychlejšího přenosu.
- Timeout a Transfer Size Option[4] – Dovoluje zvolení timeoutu a zaslání velikosti souboru serveru při Write Requestu (a tím zjištění, zda je na serveru dostatek místa pro tento soubor), nebo získání informace o velikosti souboru od serveru při Read Requestu.

S implementací výše zmíněných rozšíření souvisí i další typ packetu:

- OACK packet – slouží k potvrzení voleb zaslaných klientem, odesílá server

2 Návrh aplikace

Aplikace byla vyvíjena v jazyce C++. Vzhledem k tomu, že je to objektově orientovaný programovací jazyk, byla při návrhu aplikace rozdělena do několika samostatných částí (tříd), jelikož jak klientská, tak serverová část používají více stejných věcí. Proto jsem se snažila o co největší abstrakci a znovupoužití kódu.

[1] User Datagram Protokol, RFC 768, viz `https://www.ietf.org/rfc/rfc768.txt`

[2] TFTP Option Extension, RFC 2347, viz `https://datatracker.ietf.org/doc/html/rfc2347`

[3] TFTP Blocksize Option, RFC 2348, viz `https://datatracker.ietf.org/doc/html/rfc2348`

[4] TFTP Timeout Interval and Transfer Size Options, RFC 2349, viz `https://datatracker.ietf.org/doc/html/rfc2349`

- Třída `Client` – Implementován dle návrhového vzoru Singleton. Obstarává zpracování argumentů příkazové řádky při spuštění klienta, základní navázání komunikace se serverem a samotný přenos dat pomocí stavového automatu.
- Třída `Server` – Implementován dle návrhového vzoru Singleton. Obsahuje zpracování argumentů příkazové řádky při spuštění serveru, inicializaci poslechu na výchozím nebo zvoleném portu, přijímání nových klientů a správu vláken obsluhujících klienty.
- Třída `ClientHandler` – Řídí celý přenos dat pomocí stavového automatu, komunikuje s klientem – pro každého klienta je vytvořeno jedno vlákno.
- Třída `TFTPPacket` – Třída pro správu packetů – jejich vytváření, zpracování přijatých, odesílání. Dědí z ní všechny podtřídy packetů – `RRQWRQPacket`, `DATAPacket`, `ACKPacket`, `OACKPacket`, `ERRORPacket`.
- Třída `OutputHandler` – Slouží pro výpisy na standardní výstup a standardní chybový výstup, aby v případě připojení více klientů vždy na `stderr` a `stdout` vypisovalo pouze jedno vlákno.
- Pro činnosti nebo algoritmy prováděné na více místech, které, kdyby se nacházely přímo v kódu, by kód znepřehledňovaly, byl vytvořen soubor `helper_functions`.

3 Implementace

3.1 Klientská část

Klient obsahuje všechny základní funkce spojené s navázáním komunikace se serverem a přenos dat.

Po předání a zpracování argumentů příkazové řádky třídě `Client` se přenos zahájí voláním metody `communicate`. Ta zařídí vytvoření socketu a všech věcí spojených s přenosem. Samotný přenos je implementován konečným stavovým automatem, který řídí odesílání a příjem packetů. Pro práci s packety (vytváření, odesílání, zpracování příchozích) se využívá třídy `TFTPPacket` a tříd z ní dědících. Každý typ packetu (dle opcode) má svou vlastní podtřídu za účelem generalizace metod a odstínění klienta od jednotlivých implementačních detailů při zpracování každého packetu. Samotná třída TFTPPacket pak implementuje metodu `parsePacket`, která vrací právě podtřídu typu packetu, který byl přijat. To umožní klientovi jednoduše kontrolovat, zda přijatý packet je typu, který klient očekával, či nikoli, bez toho, aby si musel sám extrahovat z přijaté zprávy opcode a porovnávat ho. Tato metoda rovněž zapíše do atributů třídy přijatého packetu informace, které byly doručeny ve zprávě. Pokud je klientovi doručen packet, který klient neočekával, klient zašle serveru chybovou zprávu a ukončí se. Podobně klient také reaguje na signál SIGINT. U klienta bylo implementováno rozšíření blocksize a timeout (hlavně z důvodu zkoušení funkcí serveru), ale nyní je nastaven tak, že při iniciaci spojení žádné volby do packetu nepřidává, u sebe má nastavenou blocksize na 512 bytů (výchozí hodnota podle základního RFC TFTP přenosu).Timeout je nastaven na 2 sekundy s tím, že pokud klient neobdrží od serveru odpověď na zaslanou zprávu v daném čase, zašle ji opakovaně a při každém pokusu se timeout exponenciálně zvyšuje. Pokud klient neobdrží odpověď ani po celkovém počtu 5 pokusů, klient se bez odeslání chyby serveru ukončí s tím, že spojení se serverem bylo ztraceno. Přenos dat může být tedy ukončen 4 různými situacemi: byl obdržen signál SIGINT, byl obdržen poslední packet dat, server klientovi ani po opakovaném zasílání packetů neodpověděl, nebo při přenosu nastala chyba.

Klient každý přijatý packet, který je možné zpracovat, vypisuje na standardní chybový výstup. Pokud klientovi dojde packet z portu, který nepatří serveru, klient zašle odesílateli chybovou zprávu a přenos normálně pokračuje dál. Při neúspěšném pokusu o přenos se všechny otevřené subory (při nahrávání na server) zavřou, či soubory otevřené pro zápis (při stahování ze serveru) smažou.

3.2 Serverová část

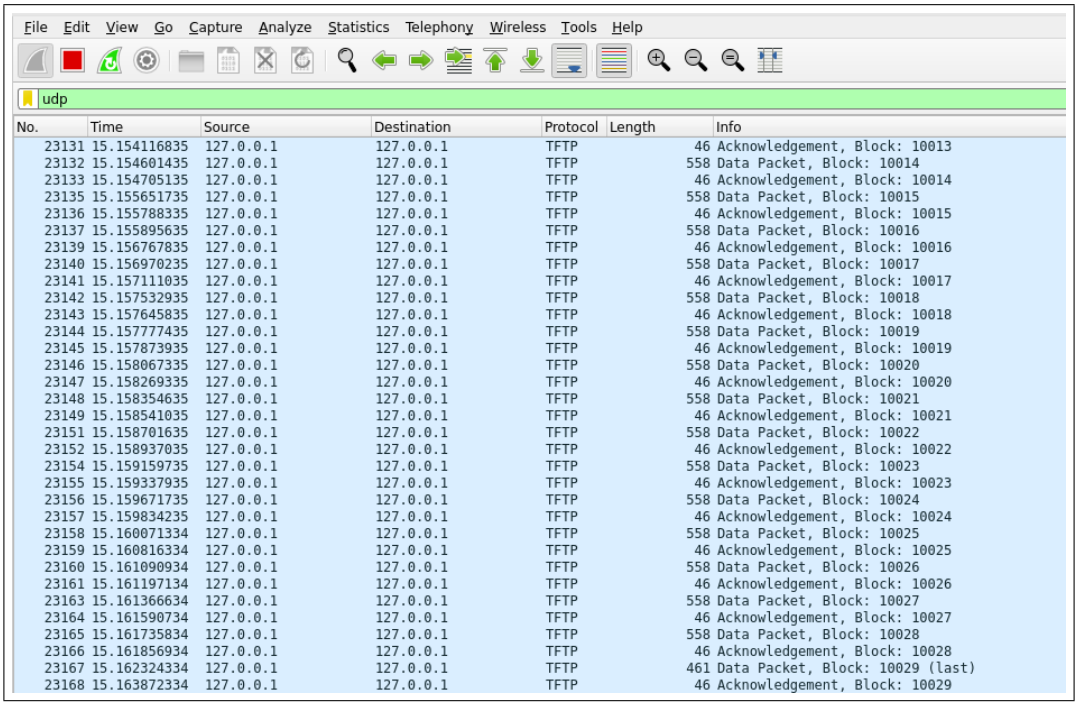
Serverová část je složena z hlavní třídy `Server` a vedlejší třídy `ClientHandler` – tyto dvě třídy dále označovány souhrnně jako *server*. V hlavní třídě server poslouchá a přijímá první zprávy od klientů. Při každé přijaté zprávě vytvoří server vlákno, do kterého předá novou instanci třídy pro komunikaci s klientem. Následně server poslouchá a přijímá případné nové klienty. Zároveň je na příjem zpráv od nových klientů nastaven timeout na 100 milisekund, po jehož uplynutí server zkontroluje, zda nebyl zaslán signál `SIGINT`, a pokud ne, znovu čeká na zprávu od případného nového klienta. Tímto je zajištěno, že při žádosti o ukončení programu nezůstane server viset na přijímání zprávy, kde by nezjistil, že se má ukončit, dokud by od nějakého klienta zprávu nedostal.

Každé vytvořené vlákno je posláno do metody `ClientHandler::handleClient`, kde vytvoří nové spojení a destinaci pro zasílání zpráv nastaví dle portu klienta, ze kterého přišla původní zpráva. Následně probíhá zpracování zprávy, kterou klient inicioval spojení se serverem. Server podporuje rozšíření options, blocksize a timeout a tsize. Pokud byla v packetu s žádostí o přenos některá z těchto voleb, server zkontroluje jejich hodnotu. Pokud se vyskytuje v packetu v části pro hodnotu jakýkoli nečíselný znak, server ukončí přenos chybou označující neplatný formát volby. Pokud je hodnota kladné celé čislo, ale je pouze větší, než ta, kterou server maximálně podporuje (u blocksize je maximální hodnota dána v závislosti na maximální velikosti UDP packetu, platné hodnoty timeoutu jsou dle RFC 1 až 255 sekund včetně, tsize je vysvětleno níže), jednoduše ji server vynechá z packetu potvrzujícího volby klienta a dále používá výchozí hodnotu (u blocksize dle RFC 512 bytů, u timeoutu má server výchozí hodnotu nastavenou na 2 sekundy). Přenos je implementován tak, že po přenosu packetu s číslem bloku 65535 (největší možné číslo, které se vleze do dvou bytů, což je velikost pro číslo bloku) se blocksize nechá přetéct a pokračuje se od začátku. Tím je zajištěna možnost přenosu větších souborů bez nutnosti specifikovat například větší velikost bloku. S tím také souvisí maximální hodnota tsize. Pokud by nebylo dovoleno po dosažení maximální hodnoty čísla bloku začít počítat od začátku, maximální hodnota tsize by se rovnala maximálnímu číslu bloku násobeného maximální hodnotou velikosti bloku. Takto je maximální velikost tsize teoreticky nedefinována, ale v implementaci byla nastavena jako maximální hodnota typu `int64_t`, což je typ, do kterého je hodnota tsize uložena.

Po úspěšném zpracování prvotní zprávy od klienta je zahájen přenos. Ten je podobně jako u klienta implementován konečným stavovým automatem, který celý přenos dat řídí. Podobně jako u klienta každé vlákno během přenosu pravidelně kontroluje, zda nedošlo k obdržení `SIGINT`. Pokud ano, hlavní část serveru pro přijímání klientů čeká na všechny vlákna, než se ukončí, a poté se ukončí též. Proto server může ještě chvíli po obdržení signálu pro ukončení čekat, než vyprší timeout pro obdržení zprávy, pokud zrovna s nějakým vláknem klient nekomunikuje, vlákno zašle klientovi chybový packet oznamující ukončení serveru a poté se ukončí.

4 Testování

Pro testování bylo využito komunikace mezi implementovaným serverem a klientem a jinými již funkčními TFTP klienty a servery. Přenos souborů a chování v různých situacích byly sledovány pomocí programu Wireshark.



Obrázek 1: Ukázka komunikace mezi klientem a serverem v programu Wireshark

5 Základní informace o programu

Projekt byl psán v C++ s využitím objektově orientovaného programování a implementací některých návrhových vzorů. Verze C++ používané při překladu programu je C++17. Bylo využito standardních knihoven. Kód byl okomentován ve tvaru pro Doxygen. Při implementaci bylo využito konečných stavových automatů a vícevláknového zpracování.

6 Návod na použití

Program se překládá pomocí příkazu `make`. Výsledkem překladu jsou 2 spustitelné soubory, `tftp-client` a `tftp-server`.

6.1 Spouštění klienta

Klienta lze spustit příkazem `./tftp-client -h hostname [-p port] [-f filepath] -t dest_filepath`, kde:

- `-h hostname` – IP adresa nebo doménový název vzdáleného serveru.
- `-p port` – nepovinný argument, port, na kterém kontaktuje klient server při pokusu o navázání komunikace. Pokud není zadán, používá se výchozí port 69.
- `-f filepath` – cesta ke stahovanému souboru na serveru (download), pokud není zadán, bere se obsah stdin (upload).
- `-t dest_filepath` – cesta, pod kterou bude soubor na klientu/serveru uložen.

6.2 Spouštění serveru

Server lze spustit příkazem `./tftp-server [-p port] root_dirpath`, kde:

- `-p port` – nepovinný argument, port, na kterém bude server přijímat klienty. Pokud není zadán, používá se výchozí port 69.
- `root_dirpath` – cesta k adresáři, do kterého se budou ukládat nebo se z něj budou číst soubory.

7 Literatura

- Sollins, K., "The TFTP Protocol (Revision 2)", STD 33, RFC 1350 (viz `https://datatracker.ietf.org/doc/html/rfc1350`), MIT, July 1992.
- Malkin, G., and A. Harkin, "TFTP Option Extension", RFC 2347 (viz `https://datatracker.ietf.org/doc/html/rfc2347`), May 1998.
- Malkin, G., and A. Harkin, "TFTP Blocksize Option", RFC 2348 (viz `https://datatracker.ietf.org/doc/html/rfc2348`), May 1998.
- Malkin, G., and A. Harkin, "TFTP Timeout Interval and Transfer Size Options", RFC 2349 (viz `https://datatracker.ietf.org/doc/html/rfc2349`), May 1998.
- Postel, J., "User Datagram Protocol", RFC 768 (viz `https://www.ietf.org/rfc/rfc768.txt`, August 1980.