

ISA - Projekt

TFTP Klient + Server

Obsah

1	Úvod	2
1.1	Popis přenosu	2
1.2	Ukončení přenosu	2
1.3	Rozšíření - RFC 2347	2
1.4	Blocksize Option - RFC 2348	2
1.5	Timeout and Transfer size option - RFC 2349	2
2	Návrh aplikace	3
2.1	Klient	3
2.2	Server	3
2.3	Sdílená část	3
3	Popis implementace	3
3.1	Klient	3
3.1.1	src/client/main.cpp	3
3.1.2	src/client/tftp_client.cpp	3
3.2	Server	4
3.2.1	src/server/main.cpp	4
3.2.2	src/server/tftp_server.cpp	4
3.3	Session, Packet	4
3.3.1	Session	4
3.3.2	ClientSession	4
3.3.3	ServerSession	4
3.3.4	Packet	5
4	Informace o programu	5
4.1	Sestavení	5
4.2	Příklad použití	5
4.2.1	Klient	5
5	Testování	6
6	Literatura	6

1 Úvod

V tomto projektu jsem se zabýval implementací klientské a serverové aplikace, která umožňuje přenos souborů mezi dvěma zařízeními v síti pomocí protokolu TFTP (Trivial File Transfer Protocol). TFTP je velmi jednoduchý protokol, který obsahuje jen základní funkce FTP (File Transfer Protocol), a tím získal své uplatnění v případech, kdy FTP je nevhodný pro svou komplikovanost např. bootování bezdiskových počítačů na síti. Implementace byla provedena dle specifikace RFC 1350 a bylo implementováno i rozšíření dle specifikace RFC 2347, 2348 a 2349. Protokol funguje nad protokolem UDP a pro zajištění správného přenosu je nezbytné, aby měl vlastní řešení spojení.

1.1 Popis přenosu

Pro zahájení přenosu klient zasílá žádost pomocí Write Request Packet pro ukládání souboru na server a nebo Read Request Packet pro stahování souboru ze serveru. Server přijme žádost a uloží si zdrojový port klienta. Server otevře socket s novým portem pro další komunikaci. Pokud se jedná o žádost pro stažení, posílá server první datový blok s číslem bloku 1. Pokud se jedná o žádost o uložení, posílá server potvrzovací paket s číslem bloku 0. Klient si po zpracování datového nebo potvrzujícího paketu uloží port nového socketu serveru. Od této chvíle klient a server mohou komunikovat jen na těchto domluvených portech. Následně dochází k přenosu souboru, protože protokol funguje nad UDP je potřeba aby každý datový paket byl potvrzen potvrzovacím paketem předtím, než dojde k odeslání dalšího datového bloku. Tím je zajištěno, že soubor je uložen a přeposlán správně.

1.2 Ukončení přenosu

Datové bloky jsou v základním režimu velké 512 bytů. Klient může tuto velikost měnit přidáním nastavení v žádosti. Pokud je datový blok menší než tato domluvená velikost, znamená to, že je soubor přenesen a přenos je ukončen. Přenos se může ukončit neúspěšně v případě, že dojde k chybě.

1.3 Rozšíření - RFC 2347

V základní verzi je TFTP dost omezený. Díky tomuto mechanismu můžeme do protokolu přidat nové funkčnosti. Tyto nové funkce vkládá klient do paketu pro žádost. Server následně pošle nazpět potvrzovací paket s rozšířeními, které podporuje.

1.4 Blocksize Option - RFC 2348

Toto rozšíření umožňuje dohodnutí velikosti datového bloku mezi klientem a serverem. Tato velikost datového bloku může mít nejmenší hodnotu 8 a největší hodnotu 65464. Při zvětšení datového bloku můžeme dosáhnout zrychlení přenosu.

1.5 Timeout and Transfer size option - RFC 2349

Timeout option slouží pro změnu doby, po které jsou opětovně posílány pakety, které nebyly potvrzeny např. z důvodu zpoždění paketu nebo jeho ztráty. Tato hodnota může být v rozmezí od 1 až po 255. Transfer size slouží pro informaci o velikosti souboru. Pokud klient chce uložit soubor na server, může v žádosti poslat i velikost tohoto souboru. Pokud server nemá dostatek místa na uložení tohoto souboru, pošle klientovi chybový paket. Pokud klient žádá o stažení souboru ze serveru, posílá žádost, ve které specifikuje tuto hodnotu na 0. Server mu následně odešle paket OACK s velikostí souboru, který je uložen na serveru. V případě, že klient nemá dostatek místa pro uložení tohoto souboru, posílá serveru chybový paket. Díky tomuto rozšíření se může zamezit přenosu souboru, který se ve finále nedokáže uložit buď na klientovi nebo na serveru.

2 Návrh aplikace

Rozhodl jsem se implementovat aplikaci pomocí programovacího jazyka C++, abych pomocí objektově orientovaného programování mohl abstrahovat implementaci jednotlivých modulů. Návrh aplikace pak byl jednodušší a výsledná implementace přehlednější. Při návrhu jsem se snažil najít co největší část logiky, kterou spolu sdílí klient a server a mít ji v jednom modulu, který by mohly používat obě aplikace. Tím jsem docílil dobré znovupoužitelnosti a omezil zbytečné opakování implementace.

2.1 Klient

Klient je konfigurován pomocí parametrů při spouštění a podle zadaných parametrů pracuje buď v módu pro stahování nebo ukládání souboru na serveru. Je implementován ve třídě TFTPClient, která na server podle zvoleného módu posílá žádost s cestou souboru a módem pro přenos. Dále klient může specifikovat rozšíření pro velikost datového bloku, délku času pro timeout a nebo velikost přenášeného souboru. Po odeslání žádosti vytváří instanci třídy ClientSession, která se stará o přenos dat.

2.2 Server

Server je konfigurován pomocí parametrů při spouštění a po inicializaci poslouchá na zvoleném portu a čeká příchozí žádosti klientů. Server ji implementován pomocí třídy TFTPServer, která při žádosti od klienta spustí asynchroně funkci pro obsluhu klienta. Tím je zajištěno, že server může obsluhovat více klientů zároveň. Při obsluze klienta dojde k validaci žádosti a v případě, že je v pořádku, vytváří se instance třídy ServerSession, která zajišťuje přenos souboru podle žádaného módu.

2.3 Sdílená část

Klient a server spolu sdílí implementaci vytváření a odesílání paketů. Ty jsou implementovány třídami RequestPacket, WriteRequestPacket, ReadRequestPacket, DataPacket, ACKPacket, ErrorPacket a OACKPacket, které dědí z abstraktní báze třídy Packet. Ta obsahuje hlavní metody metody parse, serialize a send. Díky této implementaci můžu jednoduše pracovat s pakety jak na klientské tak i serverové části, a obě tyto části spolu můžou sdílet implementaci.

3 Popis implementace

3.1 Klient

3.1.1 src/client/main.cpp

Tento soubor obsahuje implementaci vstupního bodu klientské aplikace. Je zde implementováno kontrolování a načítání argumentů při spuštění. V případě, že jsou všechny argumenty validní, vytváří se instance TFTPClient a podle zvoleného módu přenosu je zavolána buď funkce upload nebo download.

3.1.2 src/client/tftp_client.cpp

Implementace v tomto souboru zajišťuje vytvoření nového soketu pro komunikaci. V případě, že uživatel specifikoval hostitelské jméno, dojde k překladu tohoto jména na adresu IP. Následně klient odešle žádost na server pomocí vytvoření instance jedné z třídy RequestPacket. Po odeslání klient vytvoří instanci ClientSession, kde do konstruktoru této instance jsou zadány parametry komunikace např. datový mód (netascii, octet), cesta souboru, atd. Poté je zavolána metoda handleSession(), která zajišťuje přenos souboru.

3.2 Server

3.2.1 src/server/main.cpp

Soubor obsahuje vstupní bod serverové aplikace. Nejdříve se provede validace a načtení argumentů. V případě, že jsou argumenty chybné, je program ukončen. Jinak je vytvořena instance třídy TFTPServer s danými parametry a je zavolána metoda start().

3.2.2 src/server/tftp_server.cpp

Třída TFTPServer při zavolání svého konstruktoru vytvoří nový soket a přidělí k němu port dle parametrů při spuštění. Zároveň se otestuje, jestli cesta zadaná v parametrech existuje a pokud ne, je vytvořena. Ve složce pak server spravuje přenášené soubory. Metoda start() poslouchá na daném portu, a zpracovává žádosti od klientů. V případě že na soket přijdou data, je asynchroně zavolána funkce handleClientRequest. Server si toto asynchronní volání uloží jako tzv. feature a během své funkce si takto uchovává aktivní spojení s klienty. Při dokončení přenosu je daná feature smazána. Toto uchování slouží k správnému ukončení spojení v případě, že by se server ukončil pomocí signálu SIGINT. V takovém případě by server počkal, až by se všechny přenosy ukončily pomocí chybového paketu a následně by vyčistil všechny zdroje. Metoda handleClientRequest zpracuje žádost klienta a v případě, že je validní, vytváří instanci třídy ServerSession s potřebnými parametry. Tato instance se pak stará o přenos dat.

3.3 Session, Packet

Tyto třídy jsou abstraktními třídami pro sdílení logiky implementace mezi klientskou a serverovou částí. Z třídy Session pak dědí třídy ClientSession a ServerSession, které slouží k uchování stavu přenosu a přijímání dat z komunikace s klientem.

3.3.1 Session

Tato třída je funguje jako stavový automat, který zajišťuje správný přenos souborů. V případě že je zpracovaný paket ve špatném stavu, je odeslán chybový paket a ukončen přenos. Třída dále obsahuje proměnné nezbytné k přenosu např. velikost datového bloku, nastavený timeout atd.

3.3.2 ClientSession

Tato třída obsahuje hlavní implementaci pro přenos souboru na klientské části. V metodě handleSession() se v případě stahování otevře soubor pro zápis. Následně se pak přejde do hlavního cyklu, ve kterém jsou přijímány data ze soketu. Data jsou pak zpracovány do paketu, a v případě že dojde při zpracování k chybě, klient odešle na server chybový paket a ukončí přenos. Při úspěšném zpracování paketu je zavolána funkce handleClient, která podle toho, jaký paket došel a v jakém stavu se přenos nachází, provede akce a případně změní stav přenosu. V případě, že je přenos v chybovém stavu a nebo je přenos u konce, je ClientSession ukončena.

3.3.3 ServerSession

Podobně jako třída ClientSession, tato třída obsahuje implementaci pro přenos souborů na serverové části. V případě klientské žádosti o stahování souboru se nejdříve zkontroluje, jestli daný soubor existuje, a pokud ano, dojde k jeho otevření. Následně je načten a poslán první datový blok a změní se stav přenosu. Po odeslání prvního paketu začne přenos v hlavní smyčce přijímat pakety z komunikace s klientem. Při režimu stahování server posílá data a čeká na potvrzení od klienta. V případě, že dojde k chybě nebo byl odeslán poslední datový blok je přenos ukončen a jsou uvolněny veškeré zdroje. Při žádosti o uložení souboru se nejdříve zkontroluje, jestli daný soubor už neexistuje, pokud ne, je soubor otevřen pro zápis a klientovi je zaslán potvrzovací paket. Následně se přejde do stavu, kdy server ve

smyčce přijímá pakety z komunikace s klientem a zpracovává je. Při režimu ukládání server zpracovává data od klienta a ukládá je do souboru. Pokud při zápisu nedošlo k chybě, odesílá potvrzovací paket, v opačném případě vrací chybu, přenos je ukončen jsou uvolněny zdroje a soubor je smazán.

3.3.4 Packet

Abstraktní třída, ze které následně dědí všechny typy paketů. Tuto třídu jsem vytvořil pro abstrahování zpracování a práce s pakety. Metoda parse podle získaného operačního kódu paketu vytváří dané instance různých typů paketů, které implementují vlastní logiku metod handleClient a handleServer. Díky polymorfismu jsem pak schopný ve třídě Session pracovat jen s třídou Packet, ale chování je dáno typem zpracovaného paketu. Jednotlivé pakety, potom provádí akce podle toho, v jakém stavu se přenos nachází. Při úspěšném zpracování potom stav přenosu změní. Tím je zajištěno fungování stavového automatu.

4 Informace o programu

Program je implementován pomocí C++20, a pro sestavení je použit překladač g++. Program byl vyvíjen pomocí Apple clang version 14.0.3.

4.1 Sestavení

V kořenovém adresáři se nachází soubor Makefile, který slouží k sestavení programu. Pro sestavení stačí spustit příkaz:

```
$ make
```

Po úspěšném sestavení jsou vytvořeny v kořenové složce aplikace tftp-server a tftp-client. Při vývoji byl použit GNU Make 3.81.

Pro vyčištění sestaveného programu lze použít příkaz:

```
$ make clean
```

4.2 Příklad použití

4.2.1 Klient

Pro stáhnutí souboru ze serveru, je potřeba aplikaci spustit pomocí příkazu:

```
$/tftp-client -h <hostname> [-p port] -f <cesta-k-souboru-na-serveru> -t <cesta-pro-ulozeni>
```

- -h - hostitelské jméno nebo IP adresa serveru
- -p - port, na kterém je spuštěn server, pokud není specifikováno, tak se používá 69
- -f - cesta k souboru, který je na serveru
- -t - cesta, kam bude soubor uložen lokálně

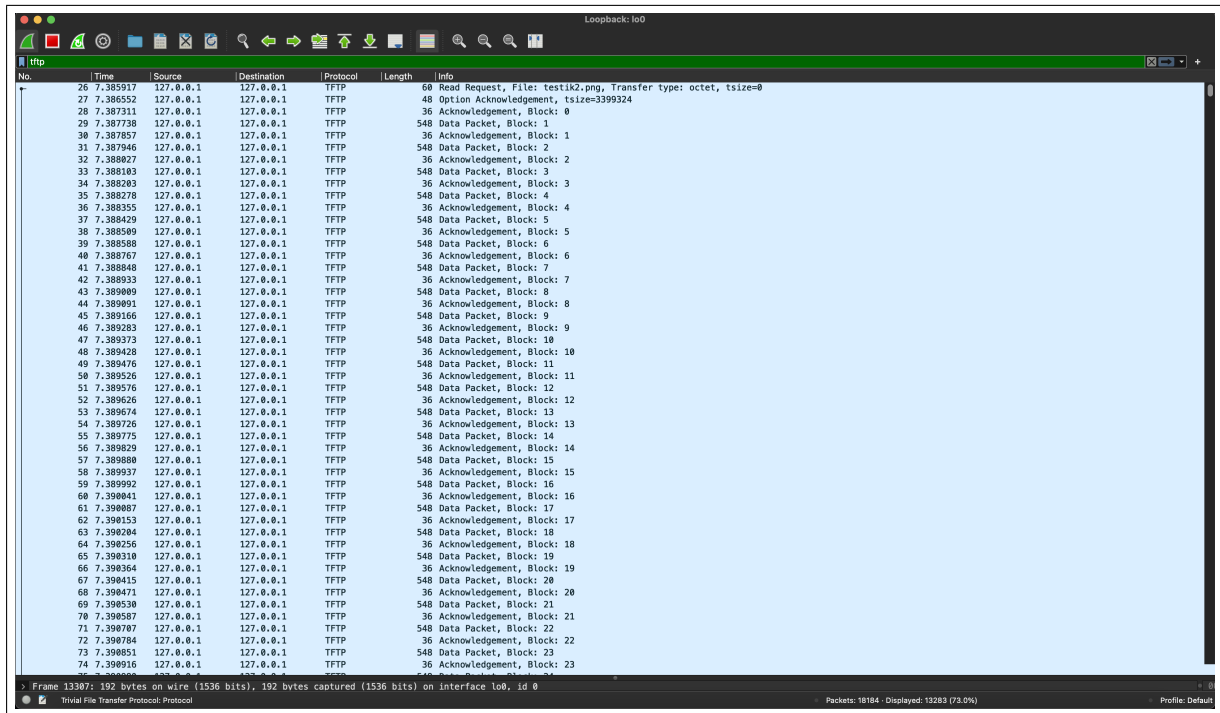
Pro uložení souboru na server, je potřeba aplikaci spustit pomocí příkazu:

```
$/tftp-client -h <hostname> [-p port] -t <cesta-k-souboru-na-serveru-pro-ulozeni>
```

- -h - hostitelské jméno nebo IP adresa serveru
- -p - port, na kterém je spuštěn server, pokud není specifikováno, tak se používá 69
- -t - cesta, kam bude soubor uložen na serveru
- v tomto módu se data berou ze standardního výstupu

5 Testování

Pro testování se v kořenovém souboru nachází program `test_tftp.py`. Tento program v programovacím jazyku Python jsem použil pro otestování mé implementace serveru. Testuji několik testovacích případů od nevalidního formátu paketů až po testování timeoutu. Díky automatickým testům jsem si mohl ověřovat svoji implementaci v případech, kdy jsem něco měnil. Testovací skript lze spustit pomocí příkazu `pytest` . . Mimo automatizovaných testů jsem prováděl i manuální testování, kdy jsem zkoušel přenést různé soubory a zjišťoval, jestli se přenesly korektně. Komunikaci jsem sledoval pomocí nástroje Wireshark viz. obrázek.



No.	Time	Source	Destination	Protocol	Length	Info
26	7.385917	127.0.0.1	127.0.0.1	TFTP	60	Read Request, File: test1k2.png, Transfer type: octet, tsize=0
27	7.386552	127.0.0.1	127.0.0.1	TFTP	48	Option Acknowledgement, tsize=3399324
28	7.387311	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 0
29	7.387738	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 1
30	7.387857	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 1
31	7.387946	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 2
32	7.388827	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 2
33	7.388183	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 3
34	7.388283	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 3
35	7.388278	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 4
36	7.388355	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 4
37	7.388429	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 5
38	7.388589	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 5
39	7.388588	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 6
40	7.388767	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 6
41	7.388848	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 7
42	7.388933	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 7
43	7.389089	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 8
44	7.389091	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 8
45	7.389166	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 9
46	7.389283	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 9
47	7.389373	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 10
48	7.389428	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 10
49	7.389476	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 11
50	7.389526	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 11
51	7.389576	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 12
52	7.389626	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 12
53	7.389674	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 13
54	7.389726	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 13
55	7.389775	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 14
56	7.389829	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 14
57	7.389888	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 15
58	7.389937	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 15
59	7.389992	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 16
60	7.390041	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 16
61	7.390087	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 17
62	7.390153	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 17
63	7.390284	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 18
64	7.390256	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 18
65	7.390318	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 19
66	7.390364	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 19
67	7.390415	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 20
68	7.390471	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 20
69	7.390538	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 21
70	7.390587	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 21
71	7.390787	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 22
72	7.390784	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 22
73	7.390851	127.0.0.1	127.0.0.1	TFTP	548	Data Packet, Block: 23
74	7.390916	127.0.0.1	127.0.0.1	TFTP	36	Acknowledgement, Block: 23

Obrázek 1: Program Wireshark

6 Literatura

- Sollins, K., "The TFTP Protocol (Revision 2)", STD 33, RFC 1350 (Online at <https://datatracker.ietf.org/doc/html/rfc1350>), MIT, July 1992.
- Malkin, G., and A. Harkin, "TFTP Option Extension", RFC 2347 (Online at <https://datatracker.ietf.org/doc/html/rfc2347>), May 1998.
- Malkin, G., and A. Harkin, "TFTP Blocksize Option", RFC 2348 (Online at <https://datatracker.ietf.org/doc/html/rfc2348>), May 1998.
- Malkin, G., and A. Harkin, "TFTP Timeout Interval and Transfer Size Options", RFC 2349 (Online at <https://datatracker.ietf.org/doc/html/rfc2349>), May 1998.