ISA

TFTP klient

Hynek Šabacký (xsabac02)

1. listopadu 2021

Obsah

Úvod

V tomto dokumentu se budeme zabývat implementací TFTP klienta. Projekt je psán v jazyce c++ ve standardu c++17. Tento jazyk byl zvolen převážně kvůli jednoduchosti práce s datovým typem std::string, slovníkovým datovým typem map a obecně jednoduchostí použití. TFTP klient je v základu poměrně jednoduchý implementovat, nejsložitější je ošetřit všechny různé krajní situace, které mohou nastat. Samotný klient využívá pro přenos paketů knihovnu *arpa/inet.h* [1].

1 Trivial File Transfer Protocol

Trivial File Transfer Protocol se používá zkráceně jako TFTP [2]. Trivial v tomto případě označuje zároveň jednoduchost protokolu, ale tím i jeho limitace. Umí pouze kopírovat soubor ze serveru a na server. Nepodporuje např. výpis adresáře nebo autentizaci uživatele. Je implementovaný tak, aby byl přenášen pomocí UDP [3].

1.1 Zahájení přenosu

Spojení začíná požadavkem od klienta. Ten je ve tvaru

2 byty	string	byte	string	byte
Opcode	Filename	0	Mode	0

Kde Opcode je operační kód, který určuje druh požadavku. Ten může být buď zápisový (2), nebo čtecí (1). Filename je název souboru s cestou. S tím souvisí, že toto pole může být proměnlivé délky a proto je ukončeno nulovým bytem. To samé se dá říct o Mode, s tím rozdílem, že ten označuje mód přenosu. Ty mohou být až tři: octet, netascii a mail. Octet zanechává data v původním stavu a nijak je nemění, netascii převádí data do ASCII s doplňkem v RFC 764 [4]. Mail mód není v mém řešení naimplementován a proto jej zde nebudu popisovat. Na první zprávu server odpovídá ze zdrojového portu, který bude do konce spojení platit jako jediný platný pro daný přenos a také si uloží port klienta, pro který platí to samé. Tyto porty platí jako TID neboli Transfer Identifiers.

1.2 Přenos

Na čtecí požadavek server odpovída prvním blokem dat ve tvaru

2 byty	2 byty	n bytů
Opcode	Block	Data

Opcode odpovídá operačnímu kódu DATA (3), Block označuje číslo odesílaného bloku (zde 0) a Data je n bytů odesílaných nebo přijímaných dat. n je v základním TFTP přenosu 512 (může být i jiné viz. 1.3 Vyjednávání možností).

Na zápisový požadavek server odpovídá ACK zprávou ve tvaru

2 byty	2 byty
Opcode	Block

Opcode je operační kód ACK (4) a Block vyjadřuje číslo potvrzovaného bloku. V tomto případě je Block 0.

Na každý DATA paket protější strana odesílá ACK paket. Jakmile ACK packet dorazí, posílá se další blok dat. V případě, že ACK nedorazí nebo dorazí ACK na již potvrzený paket, odešle se předchozí blok dat znovu. Přenos se ukončuje ve chvíli, když je přijat nebo vyslán DATA paket, který má velikost datového bloku menší než standardní délka dat v DATA paketu. V tu chvíli přijímající strana odesílá poslední ACK paket a může skončit. Doporučuje se ale, aby počkala, zda nepřijde poslední DATA blok znovu, což by znamenalo že ACK nedošel.

Když během přenosu dojde k chybě, posíla se ERROR packet

2 byty	2 byty	string	byte
Opcode	ErrCode	ErrMsg	0

Opcode se v tomto případě rovná 5, ErrCode označuje druh chyby a ErrMsg je zpráva pro uživatele, která je ukončena nulovým bytem. Jediný případ chyby, která nevede k ukončení spojení je když strana obdrží jiné TID, než dostala na začátku. V tomto případě se pouze odešle ERROR paket, ale přenos pokračuje s původním TID.

1.3 Vyjednávání možností

Pro vyjednání parametrů přenosu se používá tzv. TFTP Option Extension [5]. Funguje to na základě přidávání možností přenosu na konec iniciační zprávy. Ta pak může vypadat takto.

2 byty	string	byte	2 byty	byte	string	byte	string	byte	...
Opcode	Filename	0	Mode	0	opt1	0	value1	0	...

...

...	string	byte	string	byte
...	optN	0	valueN	0

Kde opt je název možnosti a value je její hodnota. Obě pole jsou proměnlivé velikosti a proto jsou ukončeny nulovým bytem. Množství možností je libovolné a na jejich pořadí nezáleží, ale nesmějí se opakovat. Na tuto zprávu server může odpovědět paketem OACK ve tvaru

2 byty	string	byte	string	byte	...
Opcode	opt1	0	value1	0	...

...

...	string	byte	string	byte
...	optN	0	valueN	0

Zde je Opcode OACK (6) a možnosti jsou ve stejném formátu jako u požadavku. Server může takto odpovědet jestli vyjednávání podporuje a pozná alespoň jednu možnost. U některých možností se může i rozhodnout zda nenavrhne alternativní hodnotu možnosti. Jestliže server nějakou možnost nepozná, měl by ji pouze vynechat. Možnosti vynechané serverem nesmí klient používat. Jestliže přijdou alternativní hodnoty možností, může se klient rozhodnout zda je přijme. Server potvrzuje OACK u čtení zasláním ACK s hodnotou Block 0. U zápisu potvrzuje prvním blokem dat v paketu DATA. Dále popíšu možnosti, které se vyskytují v mém řešení.

1.3.1 tsize

[6] Udává velikost souboru k přenosu. Při čtení je v požadavku hodnota 0 a velikost souboru je přijata v OACK. Server se může rozhodnout velikost odmítnout a zaslat ERROR paket.

1.3.2 blksize

[7] Říká jaká je velkost bloku dat v DATA paketu. Server hodnotu může přijmout, nebo navrhnout hodnotu, která bude nižší. Ukončení přenosu zůstává stejné. Jestliže je velikost souboru násobkem blksize, má ukončující DATA paket položku data o délce 0.

1.3.3 timeout

[6] Obsahuje dobu čekání na retransmisi nepotvrzených dat v sekundách. Jestliže klientovi přijde jína hodnota než zaslal, nastává chyba.

2 Implementace

Klient je naimplementovaný, tak že po jeho spuštění načítá příkazy z příkazové řádky, ty jsou zpracovávány a přetvářeny na čtecí či zápisové přenosy. Je strukturovaný tak, že pro příjem dat je jiná funkce než pro odesílání. To je zejména proto, že mají v hodně ohledech příliš různý průběh, než aby byly spojeny.

2.1 Načítání příkazů

Příkazy jsou načítány nekonečnou smyčkou while, ve které se vždy načítá jeden řádek. Ten je zpracováván pomocí slovníku, ve kterém se uchovává zda argument má parametr, případně jeho výchozí hodnota. Díky tomu, že není moc připuštěných argumentů se výskyt jednotlivých argumentů uchovává jako jednička či nula v 8 bitové proměnné *flags*. Pomocí ní a slovníku *flag_map* se zkontrolují hodnoty u parametrů, nebo se použijí výchozí (když to lze). Takto se naplní struktura *sock_opt*. Ta uchovává vše potřebné pro přenos a je využívána dále ostatními funkcemi. Argumenty, které jsou odesílány v rámci vyjednávání možností, jsou zatím uloženy ve slovníku *oack_map*, který bude popsán dále.

2.2 Začátek přenosu

Jakmile jsou zkontrolovány vstupní argumenty a struktura *sock_opt* je naplňena, nastavuje se *socket* pro přenos. V tuto chvíli se také převádí uživatelem zadaná IP adresa do struktury *sockaddr_in* nebo *sockaddr_in6* podle toho zda se jedná o IPv4 nebo IPv6. Díky tomu, že se jedná o spojení pomocí UDP, tak se neprovádí funkce *bind()*, což poměrně usnadňuje práci. Další krok je vytvoření paketu k odesílání. V tomto případě stačí vytvořit TFTP část, protože o zbytek paketu se stará knihovna *arpa/inet.h*. Paket je vytvořen ve formátu, který je popsán v sekci 1.1 Zahájení přenosu. V tomto případě se na paket nedá vytvořit struktura, protože v sobě obsahuje položky proměnlivé délky. Proto je paket uložen v *char* buffer* a naplňován pomocí např. *strcpy()* pro název souboru. Jestliže uživatel zadal parametry, které se vyjednávájí se serverem, jsou připojeny na konec paketu, jak je specifikováno v 1.3 Vyjednávání možností (vždy se odesílá tsize). Nakonec je paket odeslán a volá se funkce *write_transfer()* nebo *read_transfer()* podle požadavku.

2.3 Přenos

Pokračování programu je čekání na paket od serveru. Po přijetí paketu se kontroluje zda přišel OACK nebo ACK paket. V případě, že přišel OACK paket, se kontrolují přijaté možnosti. To se dělá za pomoci slovníku *oack_map*, ve kterém je uvedeno, zda klient možnost odeslal a když ano, tak s jakou hodnotou. Provede se kontrola, že přijaté hodnoty jsou přijatelné. V případě, že ne, odesílá se ERROR paket na server. Pokud vše proběhlo bez chyby, nebo server odeslal ACK paket, se pokračuje s přenosem, tak jak je uvedeno ve specifikaci (1 Trivial File Transfer Protocol). Po příjmu ERROR paketu se přenos ukončuje a čeká se na další příkaz.

2.4 Čtení a zápis souboru

Při požadavku o zápis na server, se soubor k přenosu prvně kontroluje, zda existuje a zda lze otevřít pro čtení. Poté se podle módu přenosu, buď převede do netascii, nebo se načítá jako binární data. Celý soubor se načte do kontejneru *vector<unsigned char>*, ze kterého se pak při přenosu odjímá pro každý paket. Při čtení je to jednodušší v tom, že přečtená data se ihned zapisují do souboru. V případě, že čtecí prenos selže se soubor maže. Jestliže je programu v průběhu čtení přerušen, soubor není smazán.

2.5 Ukončení přenosu

Přenos se u zápisu ukončuje po přijetí ACK s číslem bloku, který při odesílání obsahuje poslední data souboru. U čtení se končí ve chvíli, kdy se zašle paket ACK po přijetí posledního DATA paketu , ale ještě chvíli se počká, kdyby došel poslední DATA paket znovu. V tu chvíli by se ACK posílal opětovně. Jakmile se přenos ukončí, ať už úspěšně či neúspěšně, se pomocné struktury převedou do původního stavu a je uživateli nabídnuta příkazová řádka pro další příkazy.

Závěr

TFTP je poměrně jednoduchý na implementaci. Je vhodný pro jednoduché přenosy, kde není zapotřebí šifrovat přenos nebo autentizovat koncové uživatele. Komplikovanější na něm je ošetření všech chybových stavů, které mohou nastat. K problematice proměnlivých polí v paketu byl velice užitečný návod od Sumit Jha - A TFTP client implementation in C [8], kterým je základní implemetace silně ovlivněna.

Odkazy

[1] The Open Group. *arpa/inet.h - definitions for internet operations*. https://pubs.opengroup.org/onlinepubs/7908799/xns/arpainet.h.html. [Online; accessed 31-October-2021]. 1997.

[2] Karen R. Sollins. *THE TFTP PROTOCOL (REVISION 2)*. https://datatracker.ietf.org/doc/html/rfc1350. [Online; accessed 31-October-2021]. 1992.

[3] J. Postel. *User Datagram Protocol*. https://datatracker.ietf.org/doc/html/rfc768. [Online; accessed 31-October-2021]. 1980.

[4] J. Postel. *TELNET PROTOCOL SPECIFICATION*. https://datatracker.ietf.org/doc/html/rfc764. [Online; accessed 31-October-2021]. 1980.

[5] G. Malkin, A. Harkin. *TFTP Option Extension*. https://datatracker.ietf.org/doc/html/rfc2347. [Online; accessed 31-October-2021]. 1998.

[6] G. Malkin, A. Harkin. *TFTP Timeout Interval and Transfer Size Options*. https://datatracker.ietf.org/doc/html/rfc2349. [Online; accessed 31-October-2021]. 1998.

[7] G. Malkin, A. Harkin. *TFTP Blocksize Option*. https://datatracker.ietf.org/doc/html/rfc2348. [Online; accessed 31-October-2021]. 1998.

[8] Sumit Jha. *A TFTP client implementation in C*. https://www.linkedin.com/pulse/tftp-client-implementation-c-sumit-jha. [Online; accessed 31-October-2021]. 2020.