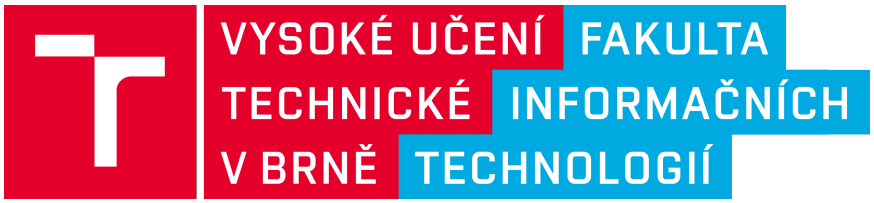


Síťové aplikace a správa sítí – Projekt

Dokumentace

TFTP Klient + Server

2023/2024 Dalibor Kříčka (xkrick01)

Obsah

1 Trivial File Transfer Protocol

1.1 TFTP

Trivial File Transfer Protocol (dále jen *TFTP*) je komunikační protokol sloužící pro přenos souborů mezi *serverem* a *klientem*. Jak již název napovídá *TFTP* je zjednodušenou verzí protokolu *FTP*. Mezi hlavní rozdíly mezi těmito protokoly patří (viz [4]):

- **TFTP:**
  - port 69 (pro data i příkazy),
  - využívá UDP,
  - nevyžaduje autentizaci,
  - spíše pro menší objemy dat.
- **FTP:**
  - porty 20 (pro data) a 21 (pro příkazy),
  - využívá TCP,
  - vyžaduje autentizaci,
  - umožňuje procházet adresáře.

1.2 Přenos dat

Protokol *TFTP* umožňuje pouze číst data ze serveru (download) nebo na server data zapisovat (upload). Pro přenos je vždy využíván jeden ze dvou režimů:

- režim *Octet* – pro přenos souborů v 8 bitovém formátu,
- režim *Netascii* – pro přenos textových souborů, kdy jsou konce řádků vždy přenášeny ve formátu *CR LF* a samostatný znak *CR* je vždy následován znakem *NULL*. Příjemce souboru si poté musí přeložit netascii data do svého vlastního formátu.

TFTP by mělo být implementováno nad prokolem UDP. Protokol UDP je nespojovaná služba, jelikož před komunikací mezi serverem a klientem nenestavá žadné spojení. Zprávy zasílá ve formě datagramů a negarantuje jejich doručení, proto je tato služba označována jako nespolehlivá. Kvůli tomu musí TFTP detekovat *duplicitní pakety*, aplikovat *časový limit* při čekání na odpověď druhé strany a poté opětovně *přeposílat* již zaslané pakety, a to právě kvůli situaci, kdy se původní paket při přenosu ztratí a nedorazí k předem určenému zažízení.

Dále musí implementovat spojení mezi klientem a serverem, aby se do přenosu nemohla zapojit žádná třetí strana a přenos tak narušit. To je realizováno pomocí *identifikátorů přenosu* (dále TID), kdy si každá strana zapamatuje port (neboli TID), ze kterého obržela první paket v daném přenosu a následně při každém dalším obrženém paketu kontroluje, zda byl zaslán správným zdrojem.

Pro řízení přenosu je využíváno 6 typů paketů, které si mezi sebou klient a server zasílají. Jedná se o následující:

- *RRQ* a *WRQ* paket – počáteční paket posílán klientem,
- *DATA* paket – paket obsahující 512 B dat (v případě možnosti *block size* jinak) a pořadové číslo datového bloku,
- *ACK* paket – paket obsahující číslo datového bloku, který potvrzuje data nebo počáteční požadavek klienta,
- *OACK* paket – potvrzuje počáteční požadavek klienta a zároveň možnosti přenosu navržené klientem,
- *ERROT* paket – obsahuje kód chyby, chybovou zprávu a předčasně ukončuje přenos.

Úspěšný přenos ukončuje DATA paket, který obsahuje méně bajtů dat než 512 B nebo než velikost bloku specifikovaná možností *block size*. Více informací k základnímu protokolu *TFTP* lze nalézt v [3].

1.3 Možnosti

Nad protokolem TFTP může být implementováno rozšíření, které umožní vyjednávání možností přenosu a jejich hodnot mezi klientem a serverem, které specifikují vybrané oblasti přenosu. Možnosti, které mohou být aplikovány na přenos:

- *Velikost bloku (Block size)* – maximální velikost datového bloku,
- *Časový limit (Timeout interval)* – čas, po který se čeká na odpověď, před znovu přeposláním předchozího paketu,
- *Velikost přenosu (Transfer size)* – velikost přenášeného souboru.

Možnosti navrhované klientem jsou zaslány v počátečním paketu (RRQ nebo WRQ), server následně rozhodne, které možnosti budou na přenos aplikovány. Možnosti, které přijme, zašle včetně jejich hodnot zpět v *OACK* paketu klientovi. Tomuto procesu se říká *vyjednávání možností* (viz [2]).

2 Návrh a implementace aplikací

Návrh aplikací klienta i serveru popsán pomocí UML diagramů aktivit viz Obrázky 1 a 2.

2.1 Chybový stav

Tyto diagramy ovšem znázorňují pouze úspěšný přenos souboru, bez ztrát paketů a bez chybových stavů. Typ paketu ERROR může být odpovědí na kterýkoliv jiný typ paketu a odesláním chybového paketu přenos končí. Ještě před ukončením po odeslaní chybového paketu odesílatel vyčká po určitý čas (ten je nastaven možností timeout nebo na výchozí hodnotu 5 vteřin), kdyby náhodou druhá strana ERROR paket neobdržela a přeposlala znovu paket, který chybu způsobil. Stejná situace nastává, když hostitel odesílá poslední ACK přenosu. Opět před ukončením přenosu vyčká daný čas, kdyby druhá strana poslední ACK neobdržela.

2.2 Duplicitní paket

Třeba je také počítat s přeposíláním paketů typu DATA a ACK, když obrdžíme duplicitní paket. Při obdržení duplicitního paketu DATA znovu přepošleme naposledy zaslaný ACK. Tak se ovšem neděje u obdržení duplicitního paketu ACK, kdy se DATA paket znovu neposílá a zdvojený ACK se ignoruje. Děje se tak, aby se předešlo jevu zvanému *Sorcerer's Apprentice Syndrome*, kdy se až do konce přenosu posílají zdvojené pakety s daty a jejich potvrzeními, a zbytečně tak narůstá režije přenosu. Více informací k tomuto problému na [1].

2.3 Adaptivní čas čekání

Klient i server implementují také funkcionalitu zvanou *exponential back-off algorithm*, kdy je po každém opakovaném vypršení časového limitu, při čekání na opověď druhé strany, čas čekání *zdvojnásoben*.

2.4 Použité technologie

Projekt byl naimplementován v jazyce C++ s využitím následujících knihoven:

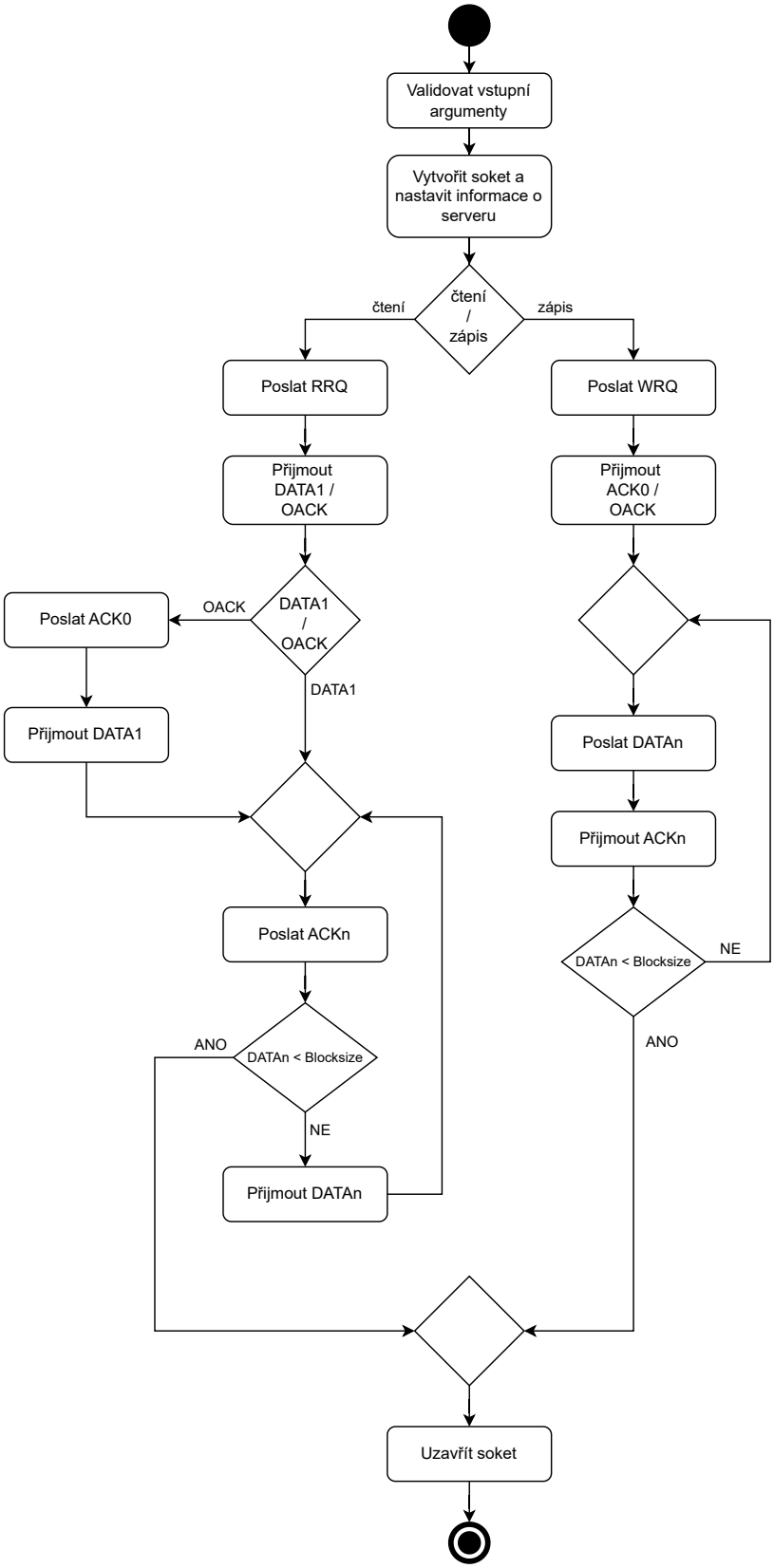
- *iostream* – definuje objekty standardního vstupu a výstupu (cout, cin),
- *string.h* – definuje funkce pro práci s řetězci typu char *,
- *regex* – definuje funkce pro práci s regulárními výrazy, využito při validaci zadaných argumentů programu,
- *sys/socket.h* a *arpa/inet.h* – poskytují funkce, datové struktury a makra pro síťovou komunikaci, využito pro správu soketů, nastavení informací o hostitelském serveru a odesílání a přijímání zpráv,
- *signal.h* – definuje funkce a makra, které umožňují reagovat na systémové signály, využito pro zachycení signálu přerušení (Ctrl-C),
- *unistd.h* – využity funkce close() pro uzavírání soketů a fork() pro vytváření paralelních procesů pro komunikaci s klienty,
- *fstream* – definuje třídu pro práci se soubory,
- *filesystem* – využito pro získání informace o volném místě na disku,
- *netdb.h* – definice pro síťové databázové operace, využito pro překlad jména hostitele na IP adresu.

2.5 Práce s daty

Každý typ paketu, který je možné při přenosu souboru poslat, je reprezentován strukturou, jejíž hodnoty reprezentují jednotlivé části paketu (například struktura packetu ACK obsahuje hodnotu operačního kódu a čísla potvrzovaného bloku). Pro každou takovou strukturu jsou poté definovány funkce na *serializace* a *deserializaci* paketu. Deserializace zpracuje proud bajtů (řetězec bajtů), který přišel od druhého hostitele a naplní strukturu příchozími daty. Serializace naopak převede strukturu na řetězec bajtů, který bude následně poslán druhému hostiteli. Díky strukturám se jednoduše operuje s daty a je umožněno lépe dosáhnout korektního chování programu podle protokolu TFTP. Struktury jsou definované ve zdrojovém souboru *tftp-packet-structures.hpp*.

2.6 Diagram aktivit – Klient

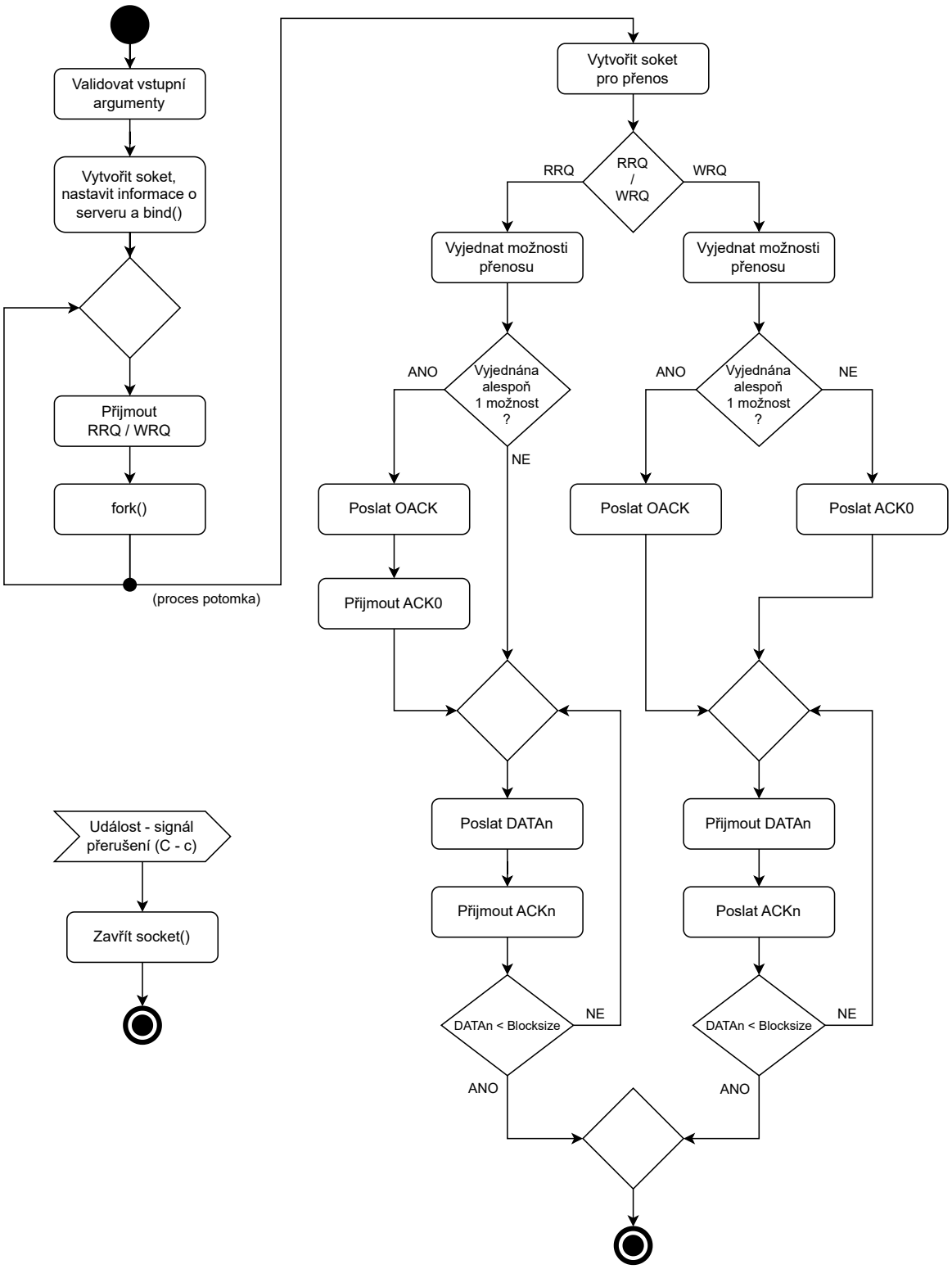
Bezchybný přenos souboru z pohledu *klienta* je znázorněn následujícím diagramem aktivit.



Obrázek 1: TFTP klient – UML diagram aktivit

2.7 Diagram aktivit – Server

Bezchybný přenos souboru z pohledu *serveru* je znázorněn následujícím diagramem aktivit.



Obrázek 2: TFTP server – UML diagram aktivit

3 Testování

Funkčnost programů byla testována pomocí referečního TFTP serveru a klienta pro systém Fedora LINUX 39. V pokročilé fázi vývoje bylo prováděno testování klienta a serveru tohoto projektu vzájemnou komunikací mezi sebou samými.

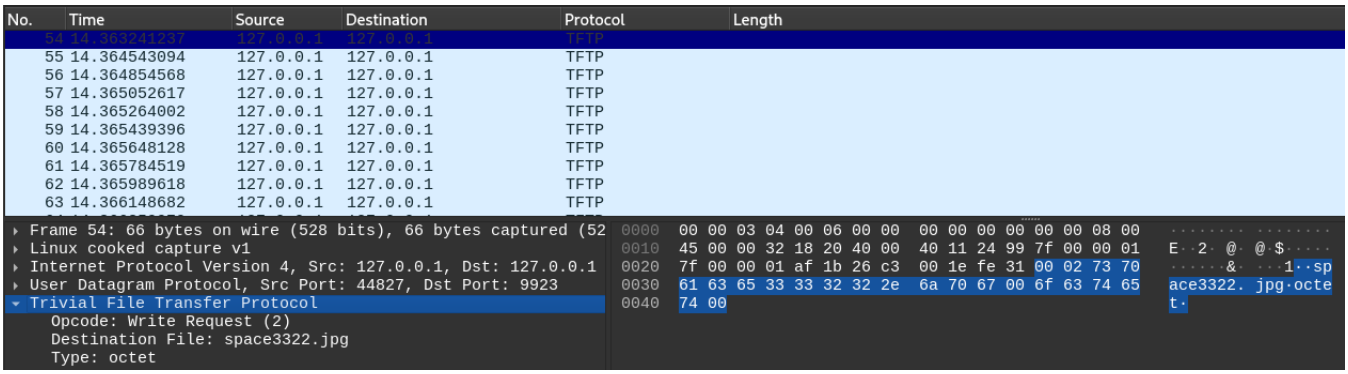
3.1 Prostředí

Programy byly testovány na následujících operačních systémech:

- NixOS – referenční vývojové prostředí pro ISA projekt,
- Fedora LINUX 39.

3.2 Wireshark

Díky programu Wireshark, který slouží pro analýzu provozu v počítačové sítí, bylo možné sledovat vzájemnou komunikaci mezi serverem a klientem. Ověřit bylo možné zdrojové/cílové IP adresy a porty a samotný obsah jedlotlivých paketů zobrazených jak v textové, tak v bajtové podobě. Wireshark také umí rozpoznat, že se jedná o protokol TFTP a poskytne přehledně všechny informace o daném paketu (operační kód, číslo bloku, cílový soubor, režim, adt.) (viz obrázek 3).



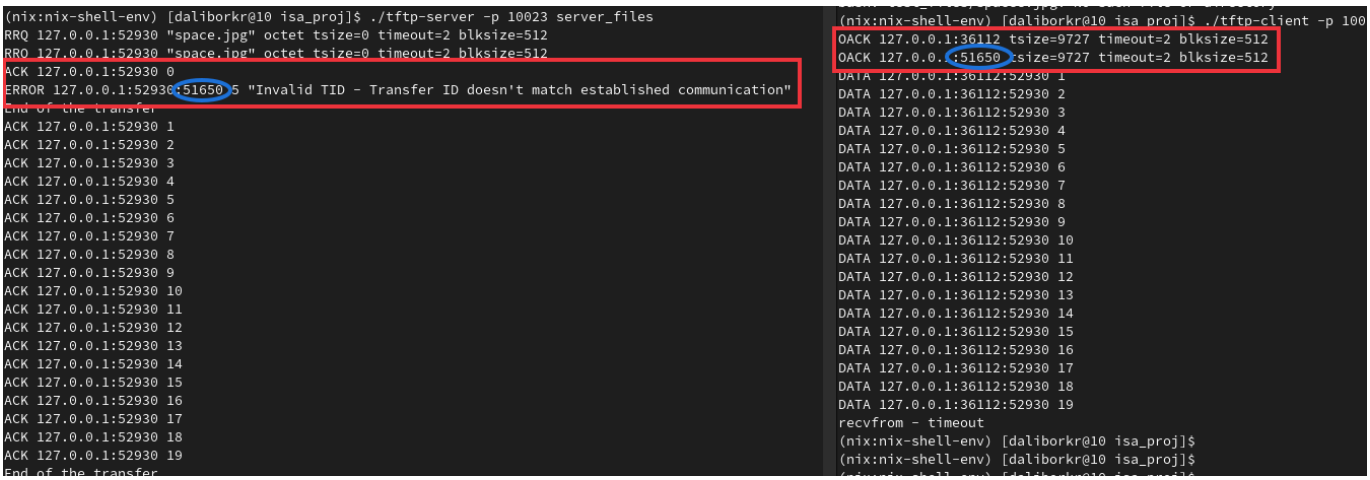
Obrázek 3: Wireshark – WRQ packet

3.3 Testování v konzoli

Spouštěním programů v konzoli bylo možné ověřit, zda výstupy programů odpovídají očekávanému výstupu, a to díky logování příchozích paketů na standardní chybový výstup. Níže na obrázcích je možné vidět chování programů v různých neobvyklých situacích, které mohou během přenosu nastat. Aby k situacím došlo a mohlo být otestováno, zda se chování klienta i serveru shoduje s prokolem TFTP, musel být dočasně pozměněn přímo zdrojový kód programů.

3.3.1 Neočekávané TID

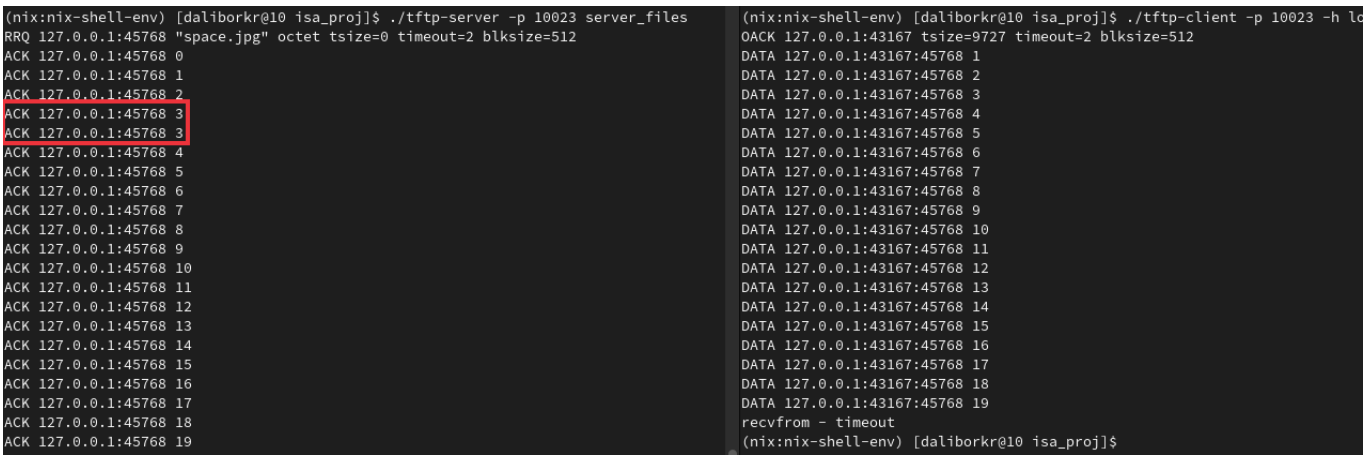
Pokud hostitel přijme paket od zdroje, který má jiné TID, než které bylo domluveno na počátku komunikace, odešle tomuto zdroji chybový paket ERROR a pokračuje v komunikaci.



Obrázek 4: Test – Neočekávané TID

3.3.2 Duplicitní ACK

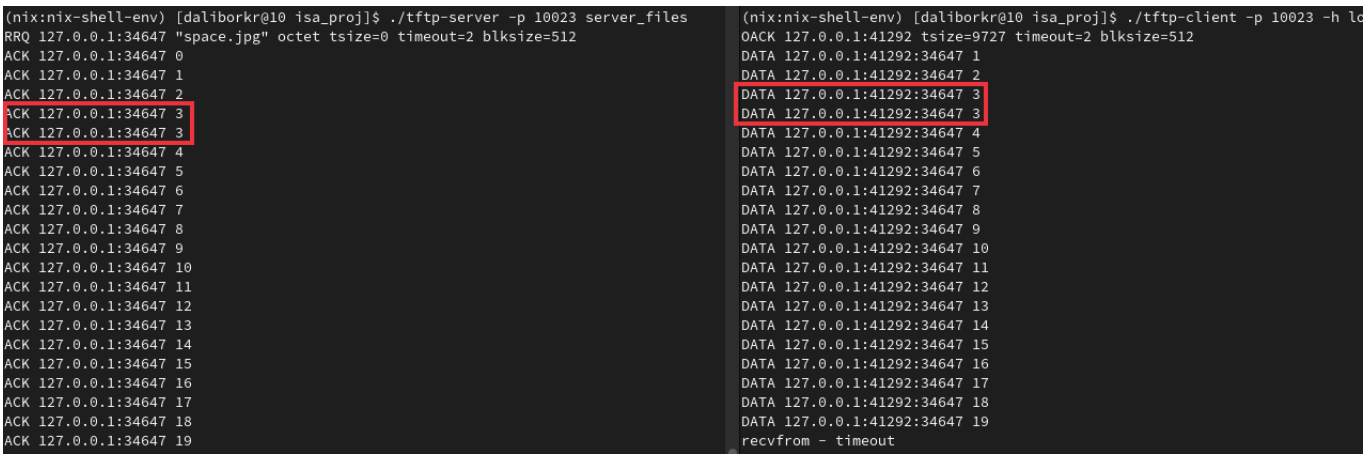
Při obdržení duplicitního ACK paketu, příjemce neposílá DATA znovu a paket ignoruje, aby nedocházelo k jevu *Sorcerer's Apprentice Syndrome* (viz 2.2).



Obrázek 5: Test – Duplicitní ACK

3.3.3 Duplicitní DATA

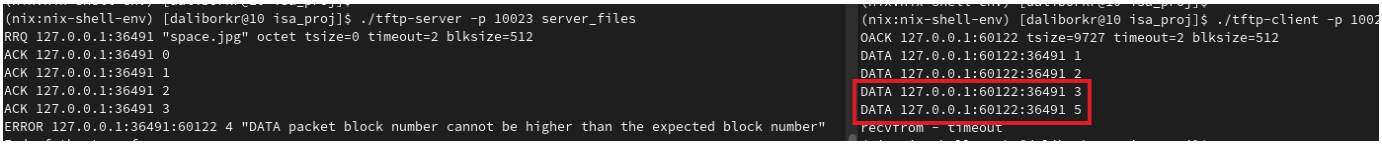
Na obržení duplicitního DATA paketu zareguje hostitel znovuposláním předchozího ACK paketu, protože se předpokládá, že byl ztracen v síti.



Obrázek 6: Test – Duplicitní DATA

3.3.4 Nekonzistentní DATA/ACK

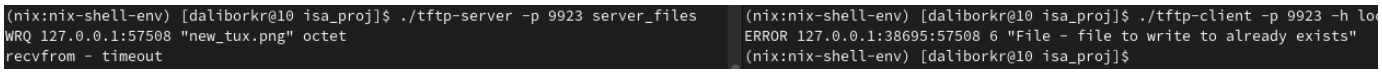
Když hostitel přijme DATA/ACK paket s číslem bloku vyšším než očekává, je vyvolán chybový stav a druhé straně je zaslán ERROR paket.



Obrázek 7: Test – Nekonzistentní DATA

3.3.5 Práce se soubory

Chybový stav vyvolá situace, když se snažíme číst soubory, které neexistují (Obrázek 8), nebo se naopak snažíme zapsat do již existujících souborů.



Obrázek 8: Test – Soubor neexistuje

4 Použití programu

4.1 Spuštění TFTP klienta

Klient je spuštěn následujícím příkazem:

```
tftp-client -h hostname [-p port] [-f filepath] -t dest_filepath
```

kde:

- *-h hostname* – je doménový název nebo IPv4 adresa serveru,
- *-p port* – je port vzdáleného serveru,
  - pokud není nastaven, tak je port roven 69,
- *-f filepath* – cesta ke stahovanému souboru na serveru (download),
  - pokud není nastavena, jedná se o nahrání obsahu stdin na server (upload),
- *-t dest_filepath* – cesta k souboru, do kterého bude přenos na serveru/lokálně uložen.

4.2 Spuštění TFTP serveru

Server je spuštěn následujícím příkazem:

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

kde:

- *-p port* – je port, na kterém se bude očekávat příchozí spojení,
  - pokud není nastaven, tak je port roven 69,
- *root_dirpath* – cesta k adresáři serveru, do kterého se budou nahrávat soubory a odkud se také budou stahovat.

4.3 Nápověda

Nápovědy k programům je v konzoli možné vypsat následujícím způsobem:

```
tftp-client --help
tftp-server --help
```

Literatura

[1] Braden, R. T.: Requirements for Internet Hosts - Application and Support. RFC 1123, Říjen 1989, doi:10.17487/RFC1123, [vid. 2023-11-15]. Dostupné z: https://www.rfc-editor.org/info/rfc1123

[2] Malkin, G. S.; Harkin, A.: TFTP Option Extension. RFC 2347, Květen 1998, doi:10.17487/RFC2347, [vid. 2023-11-15]. Dostupné z: https://www.rfc-editor.org/info/rfc2347

[3] Sollins, D. K. R.: The TFTP Protocol (Revision 2). RFC 1350, Červenec 1992, doi:10.17487/RFC1350, [vid. 2023-11-15]. Dostupné z: https://www.rfc-editor.org/info/rfc1350

[4] TAWDE, S.: FTP vs TFTP. [online], rev. 19. srpen 2023, [vid. 2023-11-15]. Dostupné z: https://www.educba.com/ftp-vs-tftp/