FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
SLOVENSKEJ TECHNICKEJ UNIVERZITY V BRATISLAVE

Komunikácia s využitím UDP protokolu

Počítačové a komunikačné siete

Zadanie 2

Erik Matovič, ID: 98347

Cvičenie: Piatok 10:00
Cvičiaci: Ing. Viktor Lančarič Rok:2020/2021

Obsah:

Zadanie

Navrhnite a implementujte program s použitím vlastného protokolu nad protokolom UDP(User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP. Program umožní komunikáciu dvoch účastníkov v lokálnej sieti Ethernet, teda prenos textových správ a ľubovoľného súboru medzi počítačmi(uzlami).

Program bude pozostávať z dvoch častí –vysielacej a prijímacej. Vysielací uzol pošle súbor inému uzlu v sieti. Predpokladá sa, že v sieti dochádza k stratám dát. Ak je posielaný súbor väčší, ako používateľom definovaná max. veľkosť fragmentu, vysielajúca strana rozloží súbor na menšie časti -fragmenty, ktoré pošle samostatne. Maximálnu veľkosť fragmentu musí mať používateľ možnosť nastaviť takú, aby neboli znova fragmentované na linkovej vrstve.

Ak je súbor poslaný ako postupnosť fragmentov, cieľový uzol vypíše správu o prijatí fragmentu s jeho poradím a či bol prenesený bez chýb. Po prijatí celého súboru na cieľovom uzle tento zobrazí správu o jeho prijatí a absolútnu cestu, kam bol prijatý súbor uložený.

Program musí obsahovať kontrolu chýb pri komunikácii a znovu vyžiadanie chybných fragmentov, vrátane pozitívneho aj negatívneho potvrdenia. Po prenesení prvého súboru pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 10-60s pokiaľ používateľ neukončí spojenie. Odporúčame riešiť cez vlastne definované signalizačné správy.

Program musí byť organizovaný tak, aby oba komunikujúce uzly mohli prepínať medzi funkciou vysielača a prijímača(nemusia byť obe súčasne) bez reštartu programu. Pri predvedení riešenia je podmienkou hodnotenia schopnosť doimplementovať jednoduchú funkcionalitu na cvičení.

Návrh programu a komunikačného protokolu

Protokol UDP je protokolom sieťového modelu TCP/IP pracujúci na transportnej vrstve. Zabezpečuje rýchly prenos dát pre zabezpečenie komunikácie v real time službách, kde sa kladie dôraz na rýchly prenos dát, napr. pri službách IP Telephony, či pri živom prenose audiovizuálneho média(Cisco Webex) alebo pri hovorových aplikáciách, avšak nezabezpečuje spoľahlivý prenos dát.

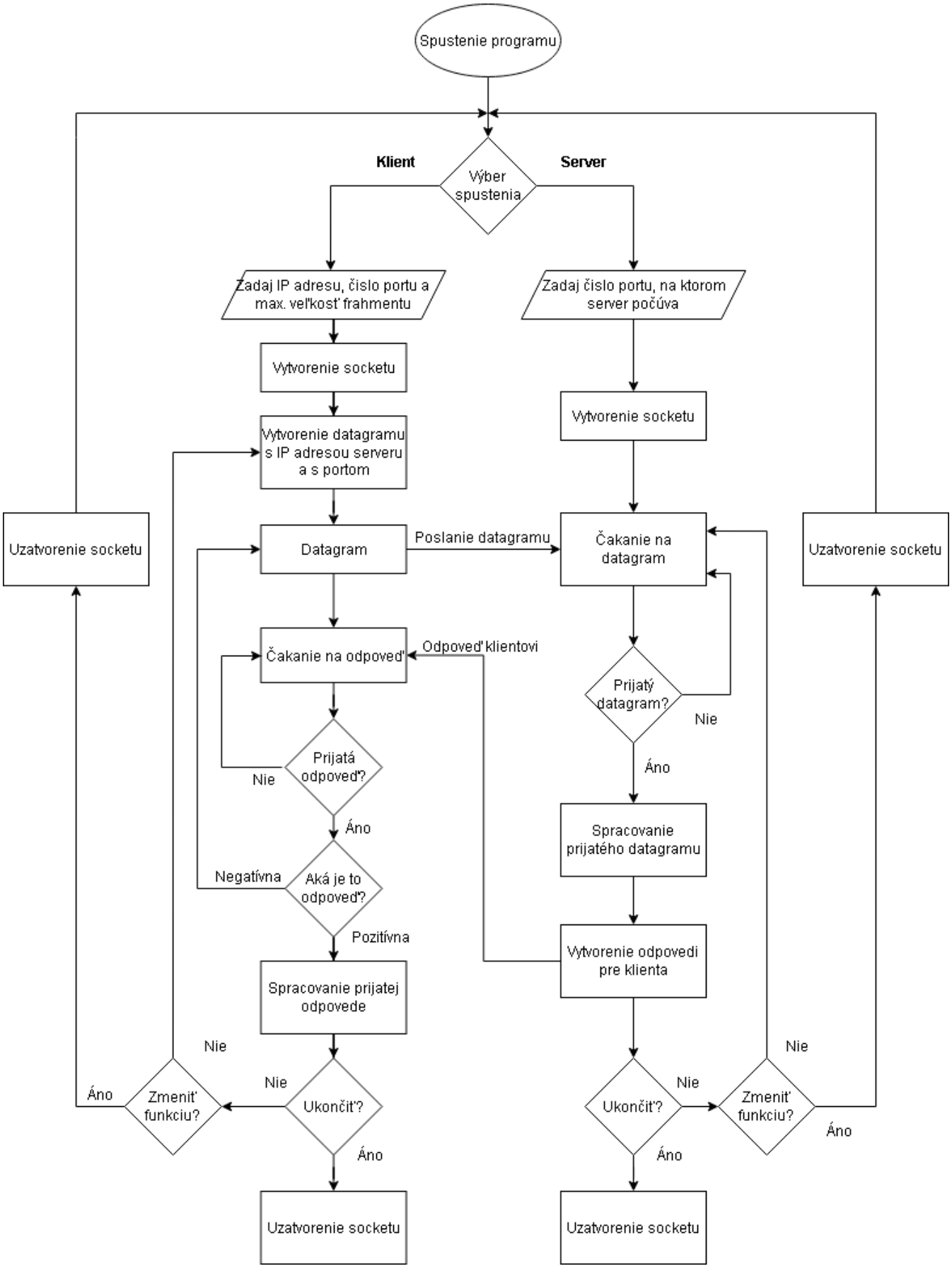
Protokol UDP je protokolom bez spojenia, ktorý, na rozdiel od protokolu TCP, znovu neposiela stratené ani poškodené dáta a zároveň nezabezpečuje zotriedenie dát. Všetky tieto vlastnosti musí preto nahrádzať navrhovaný komunikačný protokol pracujúci na aplikačnej vrstve sieťového modelu TCP/IP.

Protokol TFTP(Trivial File Transfer Protocol) je protokol pracujúci na aplikačnej vrstve sieťového modelu TCP/IP používajúci sa pri prenose súborov, pričom pracuje s protokolom UDP na transportnej vrstve. Tento protokol poslúži ako inšpirácia pri vypracovávaní zadania.

Opis návrhu

Návrh programu pozostáva z dvoch častí –vysielacej a prijímacej, t.j. klient – server model. Pri spustení programu sa vytvorí klientsky a serverový proces, pričom tieto procesy vzájomne komunikujú v lokálnej sieti Ethernet čítaním a zapisovaním do socketov.

Návrh programu pracuje so sieťovými socketmi, čo je softvérové rozhranie medzi aplikačnou a transportnou vrstvou v rámci hostiteľa, nazývané niekedy aj aplikačným programovacím rozhraním medzi aplikáciou a sieťou. Socket je vlastne pätica zložená z protokolu na transportnej vrstve, cieľovej i zdrojovej IP adresy a cieľového i zdrojového portu. Socket pri implementácií poskytuje kontrolu na aplikačnej vrstve, pričom je obmedzená kontrola na transportnej vrstve, ktorú v tomto prípade bude riadiť operačný systém.



Obr. č. 01: Diagram komunikácie aplikácie klient – server s použitím UDP

Návrh štruktúry hlavičky

Navrhovaný protokol využíva hlavičku, pozostávajúca z nasledujúcich buniek:

- Typ správy(2B) – určuje o akú správu sa jedná, napríklad:
  - 0 určuje znovu vyžiadanie dát, t.j. negatívna odpoveď
  - 1 určuje správne doručenie dát, t.j. pozitívna odpoveď
  - 2 určuje zaslanie alebo prijatie textovej správy
  - 3 určuje zaslanie alebo prijatie súboru
  - 4 určuje keepalive paket
- Sériové číslo(4B) – slúži na zoradenie správy, resp. fragmentov do správneho poradia
- Veľkosť(4B) – veľkosť prijatého alebo odoslaného fragmentu
- Počet fragmentov(4B) –celkový počet rozdelenia správy
- Kontrolný súčet(4B)
- Dáta(zvyšok)

Bit number

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Message type	Serial number
Serial number (cont.)	Size
Size (cont.)	Number of fragments
Number of fragments (cont.)	Checksum
Checksum (cont.)	Data

Obr. č. 02: Hlavička navrhovaného protokolu

Metóda kontrolnej sumy a fungovania ARQ

Ako metóda kontrolnej sumy je použitá cyklická kontrola(angl. cyclic redundancy check), skrátene CRC. Táto technika detekcie chýb je široko využívaná v dnešných počítačových sieťach a princíp fungovania je popísaný nižšie.

Pre odosielajúci uzol platí, že chce poslať d-bitový blok dát D prijímaciemu uzlu. Odosielateľ i prijímateľ majú rovnaký generátor G, t.j. bitový vzor r + 1, pričom prvý bit vzoru G je 1. Pre dátový blok D zvolí odosielateľ r-bitov R, ktoré sú pripojené k dátovému bloku D, a preto výsledný bitový vzor d + r je deliteľný bezo zvyšku číslom G pomocou modulo 2. Prijímač teda delí d + r prijatých bitov číslom G, a ak je zvyšok nulový, k chybe nedošlo, v opačnom prípade je zrejmé, že k chybe došlo.

Pre odosielateľa je nutné vybrať také R, ktoré spĺňa matematický vzťah:

$$R = D * 2^r \ mod \ G$$

Automatická žiadosť o opakovanie(angl. Automatic Repeat Request), skrátene ARQ, sa používa na riadenie chybovosti, pri ktorých vysielač od prijímača vyžaduje potvrdenie o prijatí dát. V zadaní je použitá ARQ metóda Stop & Wait, pri ktorej vysielač očakáva potvrdenie každého vyslaného paketu, pričom ďalšie pakety nie sú poslané pokiaľ nepríde potvrdenie(ACK). V prípade, že príde negatívna odpoveď alebo nepríde žiadna do určitého časového okamihu, tak vysielač opätovne vyšle paket.

Metóda pre udržanie spojenia

Po prenesení prvého súboru pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 10 až 60 sekúnd pokiaľ používateľ neukončí spojenie. Návrh hlavičky protokolu počíta s udržaním spojenia formou keepalive paketov.

Implementácia

Pri implementácií UDP socketov je využitý Python modul *socket*, ktorý umožní v rámci programu vytvárať a pracovať so sieťovými socketmi. Vloženie modulu *socket* do programu je vykonané pre obe časti programu klient – server :

$$import \ socket$$

Klient

Pre vytvorenie socketu je použitá funkcia *socket()* objektovo-orientovaným prístupom, t.j. môžeme hovoriť o metóde miesto funkcie a táto metóda sa v tomto prípade správa ako konštruktor a vráti nový objekt socketu. Funkcia v plnom tvare vyzerá nasledovne:

$socket.socket(family = AF\_INET, type = SOCK\_STREAM, proto = 0, fileno = None)$,

pričom má viacero parametrov:

- $family = AF\_INET$ adresná rodina je prvotne nastavená na IPv4 konštantou AF_INET,
- $type = SOCK\_STREAM$ typ socketu je prvotne nastavený na TCP konštantou $SOCK\_STREAM$, pre nastavenia na UDP je nutné použiť konštantu $SOCK\_DGRAM$
- $proto = 0$ číslo protokolu je zvyčajne nastavené na 0,
- $fileno = None$ je nešpecifikované, v prípade opaku sú hodnoty pre $family$, $type$ a $proto$ automaticky detegované zo špecifikácie.

V prípade pridania užívateľom nastavenej IP adresy a portu do sieťového socketu je využitá funkcia:

$$socket.sendto(bytes, address),$$

ktorá pošle do vytvoreného socketu dáta v podobe $bytes$ a v podobe `address` aj názov hostiteľa, t.j. IP adresu, s nastaveným portom v nasledujúcej podobe:

$$(cieľová\ IP\ adresa, cieľový\ port).$$

Zdrojová adresa je pripojená automaticky operačným systémom, a preto nie je potrebné explicitne písať kód pre priradenie.

Po odoslaní paketu klient čaká odpoveď od servera, ktorú je nutné zachovať v dvoch premenných – `serverData` predstavuje odpoveď servera klientovi a `serverAddress` predstavuje zdrojovú adresu paketu. Pre tieto skutočnosti je využitý nasledujúci riadok:

$$serverData, serverAddress = socket.recvfrom(bufsize[, flags]),$$

kde návratový typ $socket.recvfrom(bufsize[, flags])$ je práve pár `(bytes, address)`, pričom `bytes` predstavuje objekt reprezentujúci prijaté dáta a `address` je pár zdrojovej IP adresy a zdrojového portu, avšak pre ďalšie pracovanie si vystačíme iba s `bytes`, pretože adresu serveru máme dostupnú explicitne po zadaní užívateľom, ktorý definuje cieľovú IP adresu a cieľový port. Parameter `bufsize[, flags]` nastavuje vstupnú veľkosť vyrovnávacej pamäte.

Na záver je nutné ukončiť proces zatvorením socketu:

$$socket.close()$$

Server

Server musí byť schopný prijať dáta od klienta, a preto musí byť pripravený a spustení ako proces. Vytvorenie socketu prebieha identicky ako u klienta.

Rozdiel medzi implementáciou klienta a servera je v explicitnom viazaní serverového portu so socketom, čo zaistí, že poslanie paketu na určitý port bude smerovať do konkrétneho socketu. Dosiahnuté je to pomocou:

$$socket.bind(address),$$

pričom `address` predstavuje pár IP adresy a portu.

Server následne bude musieť byť v nekonečnej slučke, ktorá umožní prijímanie a spracovávanie prijatých paketov od klienta donekonečna, resp. pokiaľ používateľ neukončí program alebo nezmení funkcionalitu.

Prijatý paket od klienta do server socketu je spracovaný podobne ako pri klientovi:

$$clientData, clientAddress = socket.recvfrom(bufsize[, flags]),$$

pričom `clientAddress` obsahuje IP adresu klienta i číslo portu klienta. Túto informáciu server musí použiť, na rozdiel od klienta, pretože poskytuje spiatočnú adresu, t.j. server už disponuje informáciou kam poslať odpoveď. Dáta paketu sú uložené v `clientData`.

Odpoveď servera klientovi je realizované nasledovne:

$$socket.sendto(data, clientAddress\ ),$$

kde `clientAddress` disponuje informáciu kam poslať paket, táto skutočnosť je následne pripojená na `data`, t.j. samotnú odpoveď servera klientovi.

Implementácia komunikačného protokolu a klient – server komunikácie

Zadanie 2 – Komunikácia s využitím UDP protokolu – je implementované v 4 zdrojových súboroch, pričom komunikačný protokol a samotná komunikácia medzi klientom a serverom je implementovaná v nasledujúcich troch zdrojových súboroch:

- `client.py`
- `server.py`
- `protocol.py`

Klient

Implementácia komunikácie a správania klienta je definovaná v súbore `client.py`, ktorý obsahuje nasledovné správanie klienta:

- užívateľské prostredie
- nastavenie klienta
- poslanie textovej správy
- poslanie súboru
- získanie názvu súboru z cesty súboru

Server

Implementácia komunikácie a správania servera je definovaná v súbore `server.py`, ktorý obsahuje nasledovné správanie servera:

- užívateľské prostredie
- nastavenie servera
- prijatie dát
- zapísanie textovej správy na konzolu
- zapísanie súboru na vopred definovanú cestu s prijatým názvom súboru v inicializačnom pakete

Protokol

Implementácia komunikačného protokolu a jeho príslušné správanie je definované v súbore `protocol.py` , ktorý obsahuje nasledovné správanie komunikačného protokolu:

- kontrolný súčet typu CRC
- vytvorenie hlavičky pri komunikácií
- vytvorenie hlavičky pri inicializácií

Používateľské rozhranie

Program pracuje korektne vo vývojovom prostredí PyCharm Comunnity Edition 2020.2.2, ktoré opisujem v kapitole Implementačné prostredie.

Program je možné spustiť cez príkazový riadok, pričom používateľské rozhranie je samotná konzolová aplikácia. Na nasledujúcich riadkoch opíšem používateľské rozhranie, ktoré sa nelíši od operačného systému, avšak vzhľadom na fakt, že pracujem pod operačným systémom Windows 10 Home verzie 2004 (OS Build 19041.572) spoločnosti Microsoft, tak spustenie konzolovej aplikácie cez príkazový riadok popíšem na základe operačného systému Windows 10.

Pre spustenie programu cez príkazový riadok je nutné zadať nasledujúci príkaz:

`<cesta k python interpreteru> <cesta k main.py> <server/klient>`,

pričom ak je nastavená cesta k príslušnému Python interpreteru, je možné použiť v príkazovom riadku:

`python <cesta k main.py> <server/klient>`,

pričom podľa parametra server/klient je program spustený buď v režime server alebo klient, bez udania tohto argumentu je vypísaná chybová správa s následným ukončeným programom pre zamedzenie nedefinovanému správaniu.

```
def main():
    if len(sys.argv) != 2:
        print("ERROR 00: Wrong parameters!")
        sys.exit(-1)

    if sys.argv[1] == 'client':
        client.user_interface()
    elif sys.argv[1] == 'server':
        server.user_interface()
    else:
        print("ERROR 00: Wrong parameters!")
        sys.exit(-1)
```

Obr. č. 03: Implementácia spracovávania vstupných argumentov

Následne je dostupné interaktívne používateľské prostredie pre klienta i pre server.

Klient

Interaktívne používateľské prostredie klienta prebieha komunikáciou s užívateľom na úrovni konzoly. Používateľ nastaví cieľovú IP adresu servera, taktiež cieľový port server a maximálnu veľkosť fragmentu.

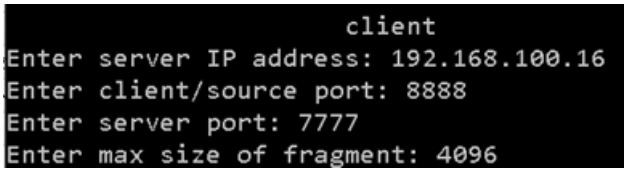
Dostupné konzolové menu na termináli umožňuje používateľovi:

- ukončiť program zadaním čísla 0
- poslať textovú správu zadaním čísla 1
- poslať súbor zadaním čísla 2
- zmeniť funkcionalitu z klienta na server zadaním čísla 9

Pred zobrazením používateľského menu klienta je nutné inicializovať, t.j. nastaviť klienta, a preto používateľ zadá na vyzvanie programu:

- serverovú, t.j. cieľovú, IP adresu v tvare IPv4
- zdrojový port klienta v rozsahu od 1024 do 65535
- serverový, t.j. cieľový, port klienta v rozsahu od 1024 do 65535
- maximálnu veľkosť fragmentu, ktorý nesmie byť menší než 1

V prípade nesprávneho vstupu užívateľa je zamedzené nedefinovanému správaniu počas behu programu.



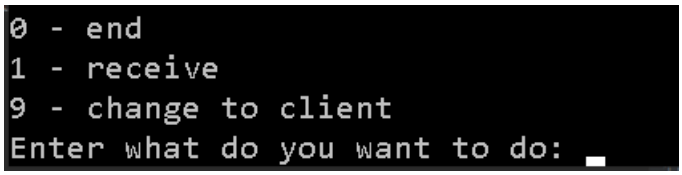
Obr. č. 04: Nastavenie klienta

Server

Interaktívne používateľské prostredie je dostupné aj pre server pre komunikáciu s užívateľom na úrovni konzoly. Používateľ nastaví zdrojový port servera, na ktorom server „počúva“(očakáva dáta) a cestu umiestnenia pre ukladania prijatých súborov.

Dostupné konzolové menu na termináli umožňuje používateľovi:

- ukončiť program zadaním čísla 0
- prijať dáta zadaním čísla 1
- zmeniť funkcionalitu zo servera na klient zadaním čísla 9



Obr. č. 05: Používateľské prostredie servera

Pred zobrazením používateľského menu serveru je nutné inicializovať, t.j. nastaviť server, a preto používateľ zadá na vyzvanie programu:

- zdrojový port serveru v rozsahu od 1024 do 65535
- existujúcu cestu k priečinku, kde server bude ukladať prijaté súbory

V prípade nesprávneho vstupu užívateľa je zamedzené nedefinovanému správaniu počas behu programu.

Obr. č. 06: Nastavenie servera

Implementačné prostredie

Pre vypracovanie zadania 2 – Komunikácia s využitím UDP protokolu – som využil možnosť programovania v programovacom jazyku Python vo vývojovom prostredí PyCharm študentskej edície Comunnity Edition s verziou 2020.2.2, ktorý umožňuje spustenie programu ako dva rozdielne procesy – pomocou terminálu a pomocou vývojového prostredia možno spustiť proces klienta a aj proces servera súčasne z jedného vývojového prostredia.

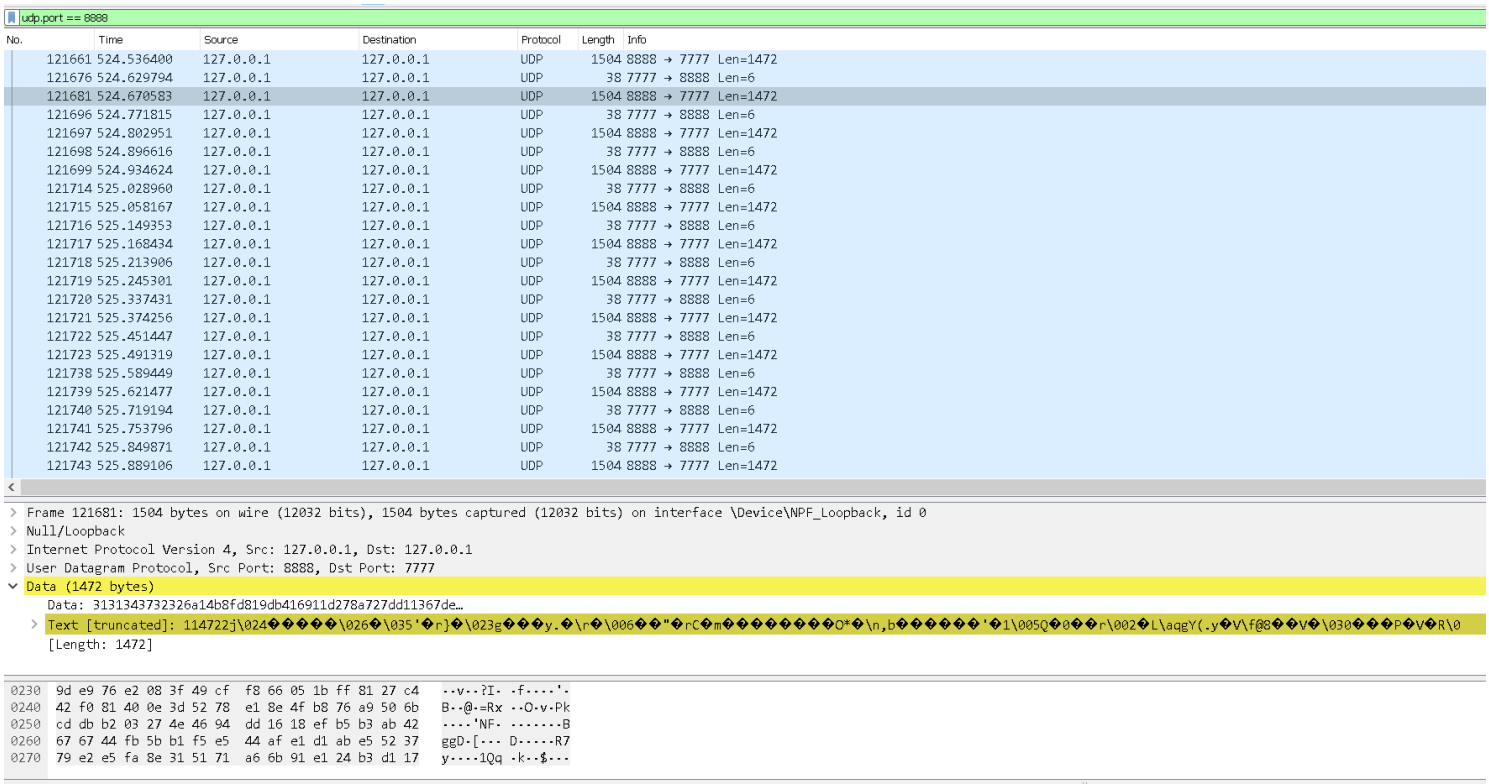
Metodika testovania

Metodika testovania zahŕňa overenie komunikácie na jednom koncovom zariadení v jednej LAN sieti, tzv. loopback, so zachyteným sieťovej premávky softvérom Wireshark posielaním textových správ a vybraných súborov.

V rámci testovania funkčnosti zadania boli odoslané po LAN sieti textové správy a nasledujúce súbory:

- zdrojové Python súbory zadania
- PDF súbory
- obrázky PNG formátu

Komunikácia bola zachytená softvérom pre analýzu sieťovej premávky – Wireshark a overená bola aj komunikácia so simuláciou chyby.



Obr. č. 07: Zachytená komunikácia klient server pri prenášaní súboru

Vlastnosti programu

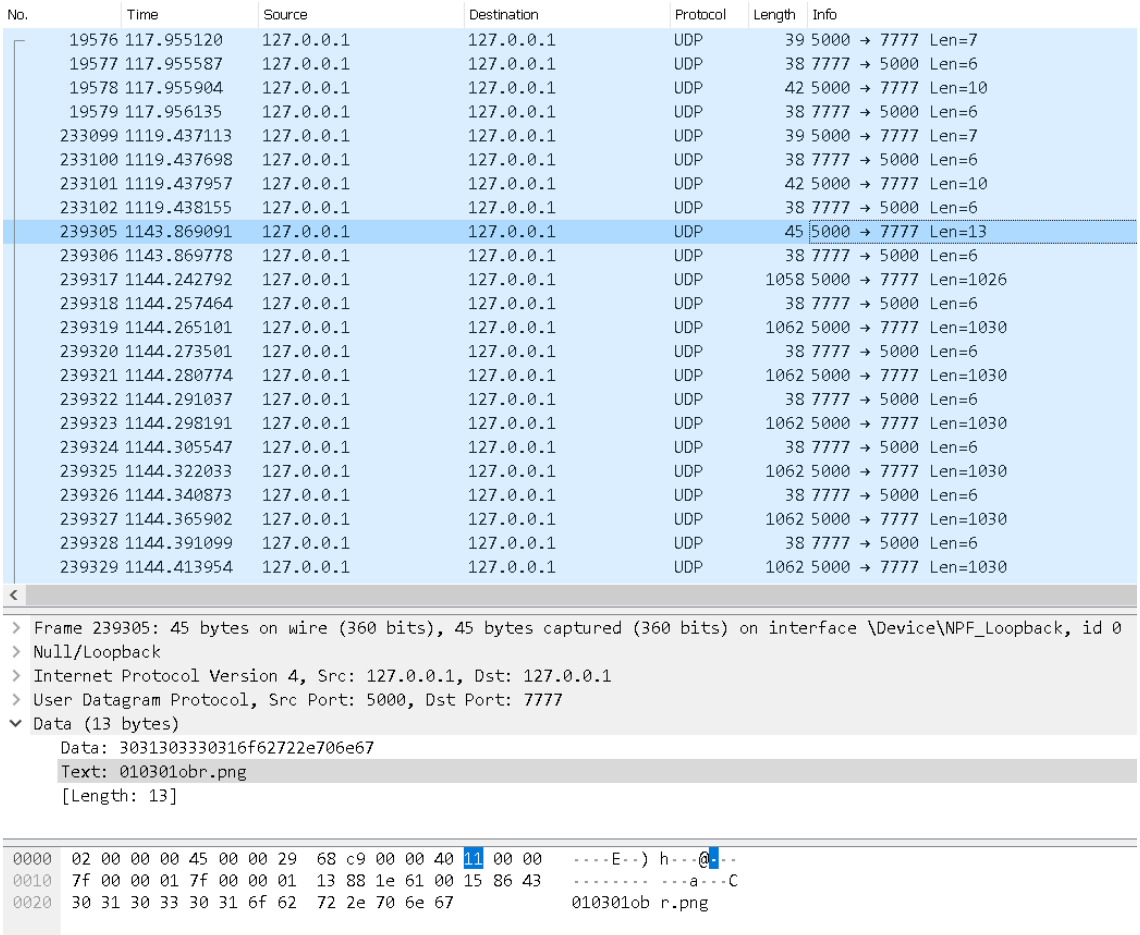
Vypracovanie zadania podliehalo niekoľkým povinným vlastnostiam, ktoré mal program spĺňať.

Nastavenie IP adresy, portu a maximálnej veľkosti fragmentácie

Program umožňuje na prijímacom uzle nastaviť prijímajúci port, na ktorom server počúva, a cestu ukladania prijatých súborov. Na vysielacom uzle je umožnené nastavenie IP adresy a portu prijímača a taktiež maximálnu veľkosť fragmentov a zdrojový port klienta pre názornú ukážku fungovania pomocou Wiresharku.

Prenos súborov

Program umožňuje server – klient komunikáciu pre prenos súborov a úspešne boli prenesené viaceré súbory vrátane menších súborov ako nastavená veľkosť fragmentu a aj súborov väčších ako nastavená veľkosť fragmentu.



Obr. č. 08: Zachytená komunikácia klient server so zvýraznením inicializačného packetu, kde ako data je poslaný názov prenášaného súboru, v tomto prípade obr.png

```
0 - end
1 - receive
9 - change to client
Enter what do you want to do: 1
Server is ready!
1395
Transfer successful, file at D:\FIIT STU\Bc\5ZS\PKS\cvika\zad2\communication_using_UDP_protocol\file_receive\2mb.pdf
```

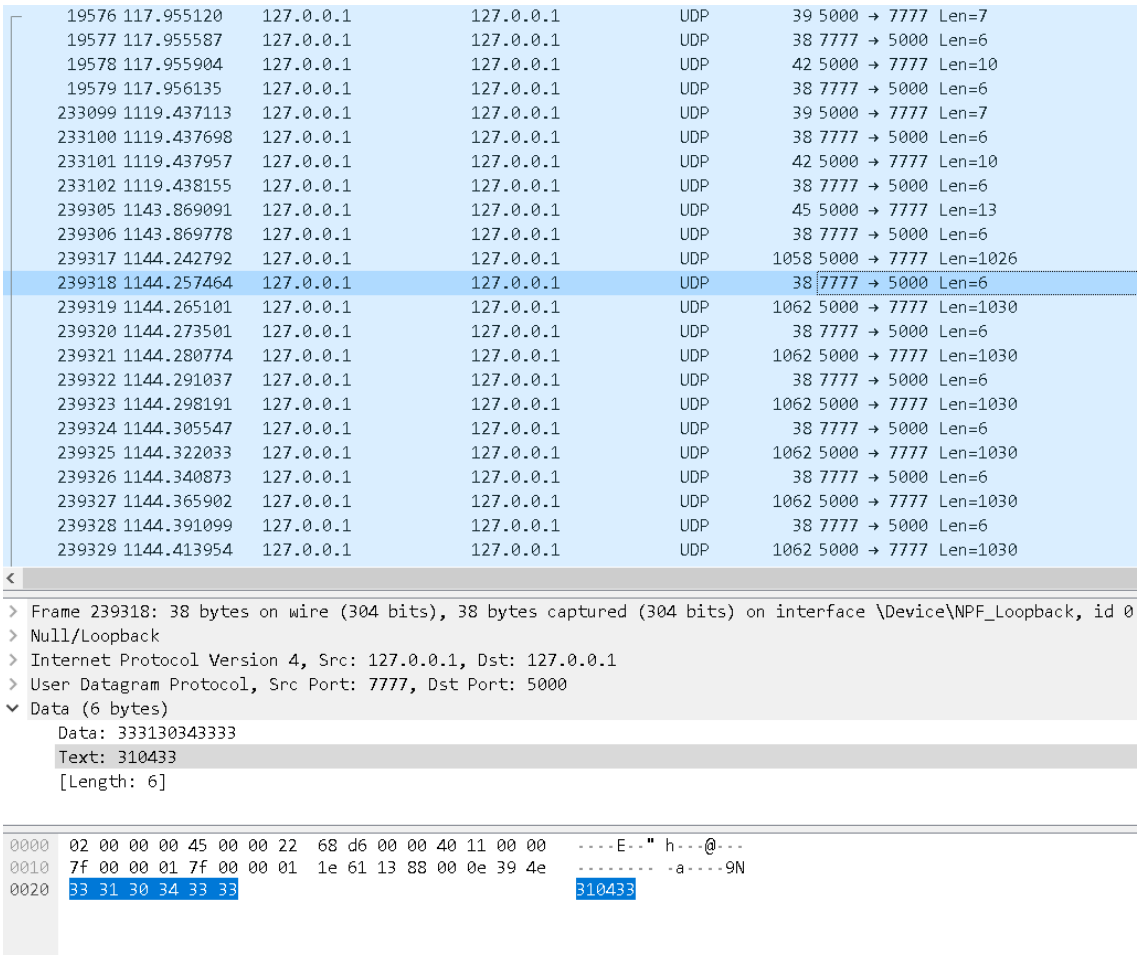
Obr. č. 09: Prijatie súboru 2mb.png na prijímači so zobrazením absolútnej cesty uloženia súboru a s počtom fragmentov(1395)

Simulácia chyby pri prenose súboru

V prípade posielania súboru sa vysielacia časť programu spýta užívateľa, či si žiada vniesť chybu pri prenose súboru pre overenie detekcie chyby a ARQ metódy.

```
Enter path to the file: files_to_send/obr.png
Make a mistake in transfer? y/n y
Initialization successful
negative acknowledgment msg
Message received!
Time: 6.465994119644165
40 40
File at D:\FIIT STU\Bc\5ZS\PKS\cvika\zad2\communication_using_UDP_protocol\files_to_send\obr.png
```

Obr. č. 10: Zadanie cesty k posielanému súboru s následnou vnesenou chybou pri prenose súboru z vysielacej časti, ktorá je užívateľovi oznámená správou `negative acknowledgment msg`, následne zobrazenie správne prijatej správy aj s celkovým počtom fragmentov rovných 40 a s nameraným časom 6 sekúnd prenosu súboru so zobrazením absolútnej cesty uloženého súboru, ktorý bol posielaný



Obr. č. 11: Odpoveď servera klientovi je zvýraznená na zachytenej komunikácií, pričom prvé číslo z hlavičky, t.j. č. 3, signalizuje znovu vyžiadanie správy(viď. Štruktúra hlavičky), pretože bola prijatá nekorešpondujúca správa

Zhodnotenie

Proces implementácie komunikačného protokolu na úrovni komunikácie medzi klientom a serverom s využitím UDP protokolu na transportnej vrstve TCP/IP modelu bol realizovaný podľa vypracovaného návrhu programu a komunikačného protokolu, avšak oproti návrhu nastali počas implementácie špecifické situácie, pri ktorých sa implementácia líši od samotného návrhu, a preto v nasledujúcich riadkoch popíšem aké rozdiely nastali medzi návrhom protokolu a samotnou implementáciou.

Opis riešenia

Podstata opisu riešenia nezmenená oproti návrhu.

Štruktúra hlavičky

Štruktúra hlavičky sa oproti pôvodnému návrhu líši a je minimalizovaná kvôli optimálnemu pracovaniu s dátami.

Implementovaný protokol využíva hlavičku, pozostávajúca z nasledujúcich buniek:

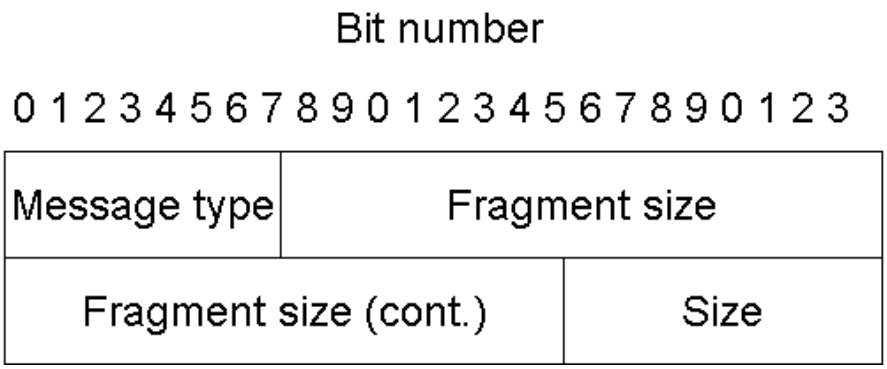
- Typ správy(1B) – zmena oproti pôvodnému návrhu vo veľkosti bajtov z dvoch bajtov minimalizovaná na jeden bajt, určuje o akú správu sa jedná a aj tu nastal zmeny oproti pôvodnému návrhu. Typy správ teraz obsahujú nasledovné signalizačné správy:
    - 0 určuje inicializáciu komunikácie pre posielanie súborov, v ktorej sa posiela názov súboru v časti data pre maximálnu optimalizáciu s dátami
    - 1 určuje dáta na zapísanie
    - 2 určuje pozitívnu odpoveď, t.j. správne prijaté dáta
    - 3 určuje negatívnu odpoveď, t.j. znovu zaslanie dát
    - 8 určuje inicializáciu textovej správy
- Veľkosť fragmentu(4B) – bezo zmeny, určuje maximálnu veľkosť prijatého/odoslaného fragmentu zadaného užívateľom, avšak hodnota nesmie byť menšia než 1 a väčšia 1466, pretože maximálna veľkosť prenášaných dát na linkovej vrstve je 1500 bajtov, pričom 8 bajtov tvorí hlavička UDP protokolu, 20 bajtov je hlavička IP protokolu a 6 bajtov je navrhnutá hlavička komunikačného protokolu, pričom podmienka fragmentácie zadania je taká, aby nedochádzalo k fragmentácií na linkovej vrstve a užívateľom definovaná hodnota fragmentácie predstavovala

maximálnu možnú hodnotu veľkosti fragmentu, oboje podmienky sú splnené.

- Kontrolný súčet(1B) – zmena oproti pôvodnému návrhu vo veľkosti bajtov zo štyroch bajtov minimalizovaná na jeden bajt.
- Dáta(zvyšok).

Pre minimalizovanie a optimalizácií práci s dátami boli vyradené nasledovné políčka z návrhu hlavičky protokolu:

- Sériové číslo(4B) – vyradené z dôvodu použitej ARQ metódy Stop & Wait, pri ktorej nie je potrebný počet fragmentov pre správne zotriedenie súborov, pretože pri nesprávne doručenom fragmente server znovu požiada o zaslanie toho istého fragmentu.
- Počet fragmentov(4B) –vyradené, pretože použitím Python modulu `select` a ARQ metódy Stop & Wait nie je potrebné prenášať informáciu o počte fragmentov, pretože spomínaný modul sa používa pre čakanie dokončenia vstupno-výstupných operácií pre možnosť ďalšej práce s dátami, avšak v počiatočnej inicializačnej správe je táto informácia prenášaná ako dáta, avšak v hlavičke nie je potrebné uchovávať túto informáciu.



Obr. č. 12: Hlavička implementovaného protokolu oproti návrhu pracuje optimálnejšie s dátami, pretože jej veľkosť je 6 bajtov oproti navrhovaným 18 bajtom v návrhu

Metóda kontrolnej sumy a fungovania ARQ

Metóda kontrolnej sumy a fungovania ARQ nezmenené, t.j. použitá je CRC metóda kontrolného súčtu a Stop & Wait metóda ARQ.

CRC metóda je inšpirovaná[1] nasledovným algoritmom:

[1] Jain, Sukirty & Chouhan, Siddharth. (2014). Cyclic Redundancy Codes: Study and Implementation. Online. https://www.researchgate.net/publication/308961643_Cyclic_Redundancy_Codes_Study_and_Implementation

1. K binárnej hodnote dát sú pridané nuly o počte dĺžky kľúča – 1
2. Využitá operácia XOR dát a kľúča pokiaľ nie sú všetky bity vykonané operáciou XOR
3. Súčet bitov posledne vykonanej operácie XOR, následnej je tento súćet vložený do hlavičky komunikačného protokolu

Ako kľúč je vybraný nasledovný polynóm:

$$x^3 + 1$$

Metóda pre udržanie spojenia

Oproti návrhu nie sú použité signalizačné správy keep alive pre udržanie spojenia, avšak miesto nich sú použité výnimky v kombinácií s inými signalizačnými správami a s Python modulom `select` použitý pre čakanie vstupno-výstupných operácií a v prípade straty spojenia je vypísané užívateľovi informácia o strate spojenia a tým aj zamedzenie nedefinovanému správaniu.