Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Zadanie 2 – Komunikácia s využitím UDP protokolu

Počítačové a komunikačné siete

Patrik Kecera
ID: 110815
ak. rok: 2020/21
Cvičiaci: Ing. Kristián Košťál PhD.

OBSAH

DOKUMENTÁCIA

1. ZADANIE ÚLOHY

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.

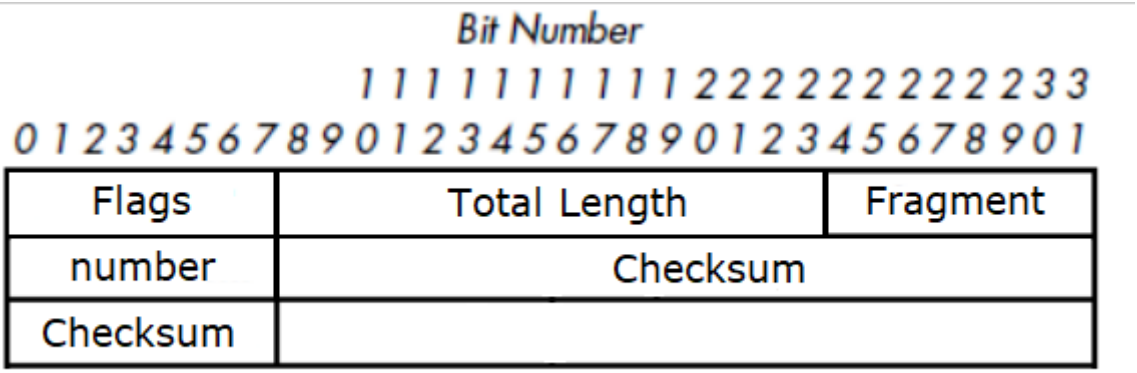
Program musí obsahovať kontrolu chýb pri komunikácii a znovu vyžiadanie chybných fragmentov, vrátane pozitívneho aj negatívneho potvrdenia. Po prenesení prvého súboru pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 5-20s pokiaľ používateľ neukončí spojenie. Odporúčame riešiť cez vlastne definované signalizačné správy.

Program musí mať nasledovné vlastnosti (minimálne):

1. Program musí byť implementovaný v jazykoch C/C++ alebo Python s využitím knižníc na prácu s UDP socket, skompilovateľný a spustiteľný v učebniach. Odporúčame použiť python modul socket, C/C++ knižnice sys/socket.h pre linux/BSD a winsock2.h pre Windows. Iné knižnice a funkcie na prácu so socketmi musia byť schválené cvičiacim. V programe môžu byť použité aj knižnice na prácu s IP adresami a portami:
   arpa/inet.h
   netinet/in.h
2. Program musí pracovať s dátami optimálne (napr. neukladať IP adresy do 4x int).
3. Pri posielaní súboru musí používateľovi umožniť určiť cieľovú IP a port.
4. Používateľ musí mať možnosť zvoliť si max. veľkosť fragmentu.
5. Obe komunikujúce strany musia byť schopné zobrazovať:
   a. názov a absolútnu cestu k súboru na danom uzle,
   b. veľkosť a počet fragmentov.
6. Možnosť simulovať chybu prenosu odoslaním minimálne 1 chybného fragmentu pri prenose súboru (do dátovej časti fragmentu je cielene vnesená chyba, to znamená, že prijímajúca strana deteguje chybu pri prenose).
7. Prijímajúca strana musí byť schopná oznámiť odosielateľovi správne aj nesprávne doručenie fragmentov. Pri nesprávnom doručení fragmentu vyžiada znovu poslať poškodené dáta.
8. Možnosť odoslať 2MB súbor a v tom prípade ich uložiť na prijímacej strane ako rovnaký súbor, pričom používateľ zadáva iba cestu k adresáru kde má byť uložený.

2. NÁVRH RIEŠENIA

2.1. Štruktúra hlavičky



Obrázok 1 štruktúra vlastnej hlavičky

Flags (1B)
Jednotlivé flagy budú jednobitové čísla. 1 označuje že je flag aktívny a 0 že je neaktívny. Flagy označujú typ paketu. Veľkosť políčka flags je 1 B ako je aj ukazané na Obrázku 1.

I - inicializačná správa (ako SYN u TCP protokolu)
K - koniec spojenia (ako FIN u TCP protokolu)
S - posielanie správy
F - posielanie súboru
U - udržiavanie spojenia tzv. keep-alive správa
A - správa o úspešnom doručení (acknowledgement)
E - doručene dáta boli chybné
Z - správa o poslaní dát znova

Flag	Označenie v dátach
I	10000000
K	01000000
S	00100000
F	00010000
U	00001000
A	00000100
E	00000010
Z	00000001

Tabuľka 1 ukážka flagov v bitoch

Total Length (2B)
Veľkosť paketu v bytoch. Podľa výpočtu môže byť maximálna dĺžka 1463 B. Výpočet: 1500B – 20B(IP) – 8B (UDP protokol) – 9 B(moja hlavička) = 1463 B.

Fragment number (2B)
Fragment number označuje poradové číslo fragmentu.

Checksum (4B)
Checksum je výsledok funkcie crc32 z knižnice zlib. Výsledok je 4 bytové nezáporne číslo.

2.2. Metóda kontrolnej sumy (checksum)

Na výpočet kontrolnej sumy (checksum) použijeme funkciu crc32 z knižnice zlib. Crc je algoritmus, ktorý vypočíta z dát kontrolnú sumu. Kontrolná suma bude slúžiť na overenie správnosti prijatého fragmentu. Bude sa porovnávať checksum v hlavičke a checksum z dát prijatého fragmentu. Funkcia crc32 funguje na delení polynómu, ktorý je nižsie. Zvyšok po delení je checksum teda kontrolná suma, ktorá sa bude posielať v hlavičke. Checksum je vždy 4 bytový unsigned int.

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

2.3. ARQ metóda

ARQ metóda, je metóda na kontrolu chýb pri prenose dát. Využívajú sa pritom správy o potvrdení a timeouty. V našom protokole budeme používať najľahšiu stop and wait ARQ metódu. Stop and wait ARQ metóda funguje nasledovne. Klient odošle fragment na server a čaká správu ack teda správu, ktorú odošle server, že úspešne prijal bezchybný fragment. Ak klient nedostane ack správu do určitého časového limitu alebo dostane inú správu od servera (správu o chybnom fragmente alebo o poslaní fragmentu znova), tak klient odošle ten istý fragment znovu.

2.4. Keep – Alive správa

Keep -Alive správu sa bude posielať od klienta ak server neposlal správu o obdržaní dát po dobu 5 sekund. Ak klient pošle 3 krát za sebou správu keep-alive a server neodpovie, tak klient odošle správu fin o ukončení spojenia. V prípade, že server obdrží správu keep-alive, tak odošle správu ack a tak sa udrží spojenie. Potom komunikácia pokračuje

2.5. Inicializácia spojenia

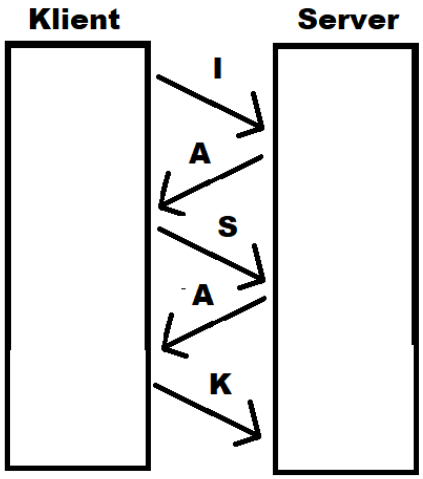
Inicializácia alebo nadviazanie spojenia funguje veľmi jednoducho. Najprv klient zadá ip adresu, port a maximálnu veľkosť fragmentu a server zadá port na ktorom počúva. Následne klient pošle serveru inicializačnú správu (flag I je aktivovaný). Server pri prijatí inicializačnej správy odošle správu ACK teda správu o úspešnom doručení (flag A je aktivovaný).

2.6. Ukončenie spojenia

Ukončenie spojenia prebieha ešte jednoduchšie ako nadviazanie. Klient odošle serveru správu o ukončení a obaja ukončia spojenie. Ukončiť spojenie sa môže buď ak server neodpovedá na keep-alive správy alebo ak klient už nechce ďalej odosielať dáta.

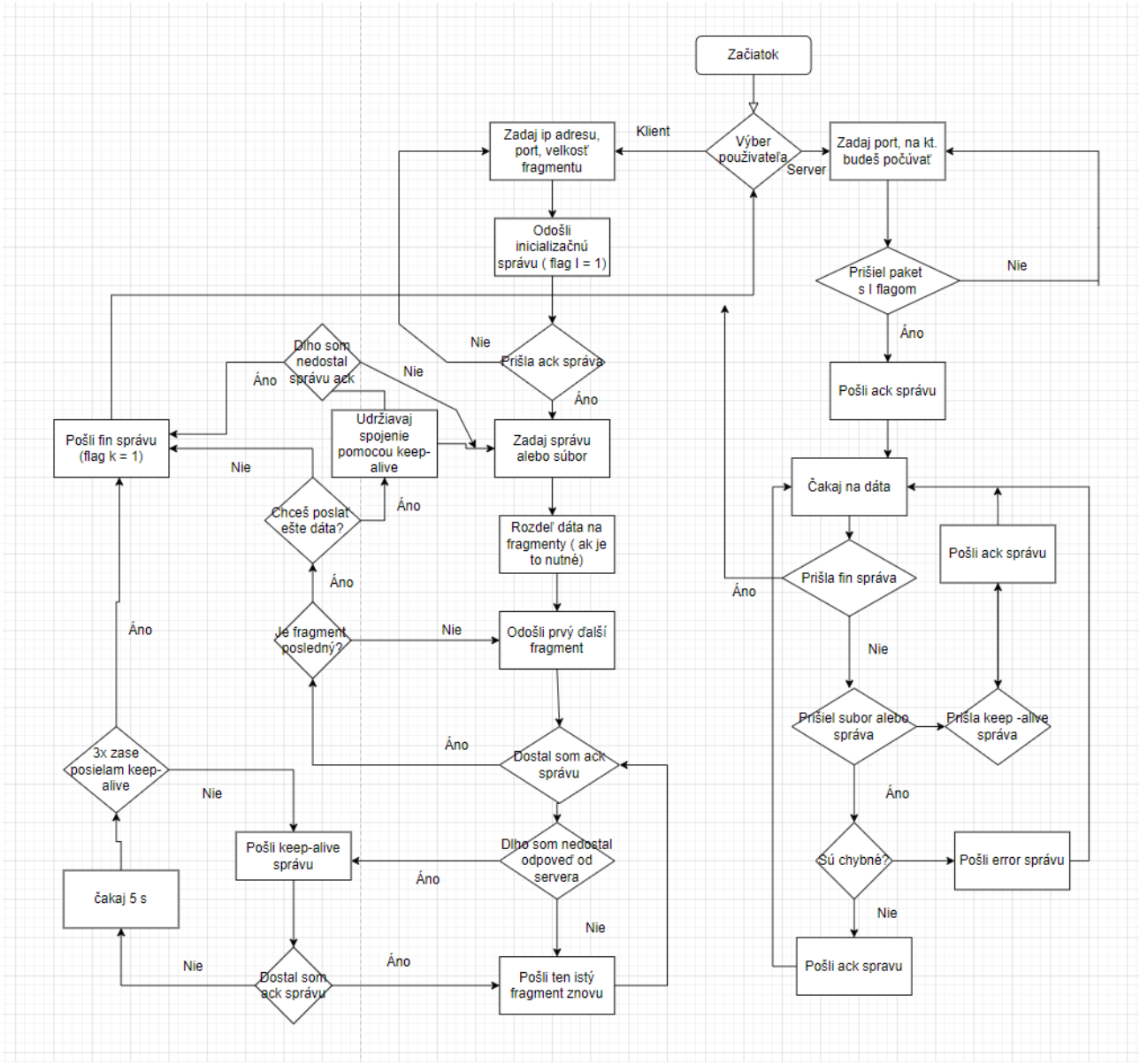
2.7. Príklad komunikácie

Na obrázku 2 je ukážka odoslania 1 správy. Najprv sa medzi klientom a serverom nadviaže spojenie pomocou inicializačnej správy a správy o úspešnom prijatí. Následne klient pošle správu a server odpovie správou o úspešnom prijatí. Ak mi nastala chyba (checksum z dát a checksum z hlavičky by sa nerovnal), tak by sa správa vyhodila a server by odoslal error správu a vyžiadal si správu znova. Nakoniec po odoslaní správ alebo po dlhom neodpovedaní servera klient odošle správu o ukončení spojenia.



Obrázok 2 ukážka odoslania 1 správy

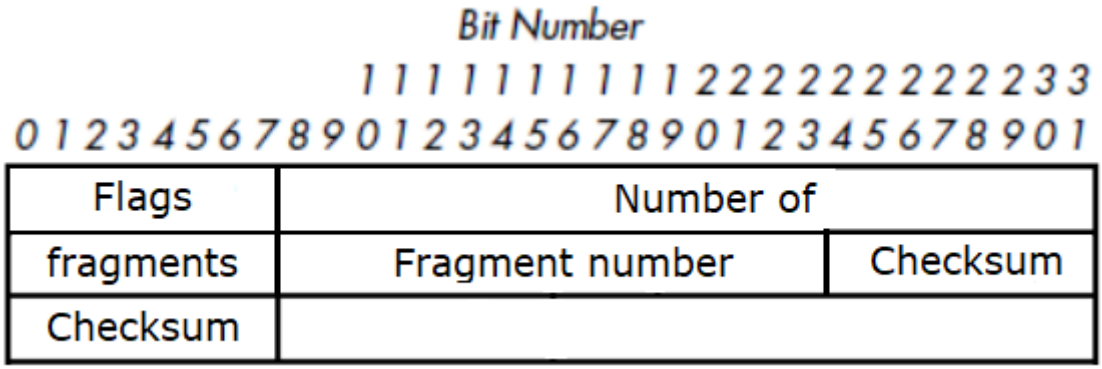
2.8. Diagram spracovávania komunikácie



Obrázok 3 Diagram spracovávania komunikácie

3. ZMENY NÁVRHU

3.1. Štruktúra hlavičky



Obrázok 4 upravená štruktúra hlavičky

V hlavičke sa zmenila veľkosť Checksumu z 4 B na 2 B. Potom sa nahradila dĺžka fragmentu počtom prenášaných fragmentov, ktoré má veľkosť 4 B.

Flags (1B)
Flagy sa zmenili minimálne. Jediná zmenená flag je Z, ktorá reprezentuje správu o vymenení rolí(server, klient)

I - inicializačná správa (ako SYN u TCP protokolu)
K - koniec spojenia (ako FIN u TCP protokolu)
S - posielanie správy
F - posielanie súboru
U - udržiavanie spojenia tzv. keep-alive správa
A - správa o úspešnom doručení (acknowledgement)
E - doručene dáta boli chybné
Z - správa o vymenení rolí

Flag	Označenie v dátach	Dekadicky
I	10000000	128
K	01000000	64
S	00100000	32
F	00010000	16
U	00001000	8
A	00000100	4
E	00000010	2
Z	00000001	1

Tabuľka 2 ukážka upravených flagov

Number of fragments (4B)
Number of fragments označuje počet fragmentov prenášaného súboru alebo správy.
4B preto aby mohlo byť prenášaných až 2^{32} fragmentov.

3.2. Metóda kontrolnej sumy (checksum)

Checsum bol upravený z 4B na 2B. Na výpočet checksumu bola použitá funkcia crc_hqx z knižnice binascii. Funkcia crc_hqx vypočíta 16 bitový crc výsledok. Na vypočítanie výsledku používa polynóm zobrazený na nasledujúcom obrázku. Táto funkcia bola vybraná lebo obsahuje minimálne zhody crc výpočtov a v porovnaní s 4 B metódou šetrí pamäť v hlavičke.

$$x^{16} + x^{12} + x^5 + 1$$

Obrázok 5 polynóm CRC-CCITT

3.3. Simulácia chyby

Simulácia chyby prebieha veľmi jednoducho. S 20 % pravdepodobnosťou je poslaný chybný fragment. Chybný fragment je taký, ktorému najprv vypočítame checksum z dát a potom dáta zmeníme.

```
if random.random() < 0.2:  # 20 % that will be send bad data
    send_data.change_data()
```

3.4. Keep-alive metóda

Keep-alive metóda bola skoro úplne zachovaná z návrhu. Rozdiel je v tom, že keep-alive metóda nefunguje počas posielania fragmentov. Keď sa posielajú fragmenty, vtedy sú nastavené timeouty a ak príde timeout, tak je správa poslaná znova. Ak je tá istá správa poslaná 3x (timeout 3x na jeden fragment), tak je klient pošle serveru správu o ukončení a vypne sa.

4. IMPLEMENTÁCIA

Zadanie bolo implementované v jazyku python 3.9. V programe boli použité knižnice zobrazené na obr. 6. Zadanie som implementoval ako jeden program, v ktorom sú 2 hlavné funkcie (server a client), preto pre komunikovanie klienta so serverom musí byť pustený program 2x na jednom PC (pri loopbacku) alebo na 2 PC, kde jeden spustí sa v programe ako server a druhý ako klient.

```
import socket
import binascii  # for crc
import threading
import math
import random
from collections import deque
import time
```

Obrázok 6 knižnice programu

V programe som implementoval aj 2 hlavné triedy. Jedna trieda sa volá fragment, ktorá reprezentuje fragment ktorý sa ide odosielať a druhá fragment_rec, ktorá reprezentuje fragment, ktorý je prijímaný.. Fragment berieme ako hlavička (9B) + prenášané dáta.

Trieda fragment

Ako bolo spomenuté, trieda fragment označuje fragment, ktorý sa ide odosielať. Táto trieda je používaná v funkcií client, pri posielaní fragmentov. Pred posielaním fragmentu sa najprv vytvorí objekt triedy fragment, ktorému sa inicializuje hlavička a pridajú sa dáta. Potom sa fragment odošle.

```
class fragment:
    def __init__(self, flag_num, num_of_frags, id_of_frag, checksum, dataa):
        self.data = bytearray()
        self.data.extend(flag_num.to_bytes(1, "big"))
        if num_of_frags is not None:
            self.data.extend(num_of_frags.to_bytes(4, "big"))
        if id_of_frag is not None:
            if id_of_frag > 65535:
                id_of_frag %= 65535
            self.data.extend(id_of_frag.to_bytes(2, "big"))
        if checksum is not None:
            self.data.extend(checksum.to_bytes(2, "big"))
        if dataa is not None:
            self.data.extend(bytearray(dataa))

    def change_data(self):
        if int.from_bytes(self.data[9:10], "big") < 200:
            self.data[9] += 1
        else:
            self.data[9] -= 1

    def get_length(self):  # returns length of fragment without header
        return len(self.data) - 9
```

Obrázok 7 trieda fragment

Trieda fragment_rec

Trieda fragment_rec označuje fragment, ktorý prichádza. Trieda fragment_rec je preto využívaná v funkcii server, keď prijíma fragmenty. Pri prijatí sa vytvorí objekt triedy fragment_rec a následne sa získajú z tohto fragmentu informácie z hlavičky.

```
class fragment_rec:
    def __init__(self, dataa):
        self.data = bytearray()
        self.data.extend(dataa)

    def get_length(self):  # returns length of fragment without header
        return len(self.data) - 9

    def get_flag(self):
        return int.from_bytes(self.data[0:1], "big")

    def get_num_of_frags(self):
        return int.from_bytes(self.data[1:5], "big")

    def get_id_of_frag(self):
        return int.from_bytes(self.data[5:7], "big")

    def get_checksum(self):
        return int.from_bytes(self.data[7:9], "big")

    def compute_checksum(self):
        return binascii.crc_hqx(self.data[9:len(self.data)], 0)

    def get_data(self):
        return self.data[9:len(self.data)]
```

Obrázok 8 trieda fragment_rec

Funkcia client

Funkcia client je volaná ak si používateľ zvolí, že chce byť klientom (chce odosielať dáta serveru). Klient si zo začiatku vypýta ip adresu, port servera a maximálnu veľkosť fragmentu (1-1465B). Potom spustí funkciu na inicializovanie spojenia. Keď sa podarí inicializovať spojenie so serverom, tak následne sa opýta používateľa akú správu má poslať serveru. Používateľ si môže zvoliť 3 typy správy M – poslanie správy, F – poslanie súboru, K – ukončenie spojenia. Potom funkcia následne podľa zvoleného príkazu zvolí čo ide odosielať sa. Pri správe používateľ zadá správu a následne ju funkcia rozdelí na fragmenty zadanej veľkosti a odosiela fragment s tým že ak je simulácia chyby úspešná, tak ešte predtým zmení dáta, aby boli chybné. Potom čaká na ACK správu od servera. Ak ju nedostane do 1 sekundy alebo ak dostane error správu, tak posiela daný fragment znova. Po odoslaní všetkých fragmentov funkcia vypíše počet správne odoslaných fragmentov a čaká 4 sekundy či sa chce server zmeniť na klient, ak ju nedostane, tak zase funkcia zobrazí akú správu má poslať.

```
while start_pos != len(message_in_bytes):
    mess_frag, end_pos = create_next_message_fragment(message_in_bytes, start_pos, max_size)
    send_mess = fragment(32, count_of_fragments, counter + 1, binascii.crc_hqx(mess_frag, 0), mess_frag)
    size_of_frag = send_mess.get_length()
    if random.random() < 0.2:  # 20 % that will be send bad data
        send_mess.change_data()
    client_soc.sendto(send_mess.data, (ip_add, port))
    client_soc.settimeout(1)
    try:
        data, app_port = client_soc.recvfrom(2048)
    except socket.timeout:
        print("Nedostal som odpoved 1 sekundu po odoslani.. Posielam znova..")
        client_soc.settimeout(None)
```

Obrázok 9 časť funkcie client

Funkcia server

Funkcia server je volaná, ak si používateľ zvolí stranu servera (prijímača). Server na začiatku si vypýta od používateľa, na ktorom porte má počúvať. V jednoduchosti potom server čaká na pakety na danom porte a ak príde nejaký paket, potom rozkóduje hlavičku a podľa flagu v nej sa rozhodne, čo robiť ďalej. Teda ak príde flag I (inicializácia spojenia), tak odošle správu ACK a inicializácia spojenia je úspešna. Pri prijímaní správy čaká na flag S a postupne si ukladá do stringu správu. Na konci keď príde posledný fragment, správu vypíše. Samozrejme pri prijímaní všetkých fragmentov je najprv porovnávaný checksum v hlavičke s vypočítaným checksumom z dát. Ak sa tieto 2 hodnoty nerovnajú, tak je fragment zahodený a je poslaná error správa. Ak sa hodnoty rovnajú, tak je poslaná ACK správa. Pri prijímaní súboru je to komplikovanejšie. Najprv musí byť prijatý paket s flagmi I a F, ktorý inicializuje prijímanie súboru. Pri tomto si server zvolí, kde sa má prijatí súbor uložiť a pošle ACK. Potom prichádzajú fragmenty súboru. Keď príde posledný fragment súboru, tak sa súbor zapíše do vybraného miesta.

Funkcia ka_func

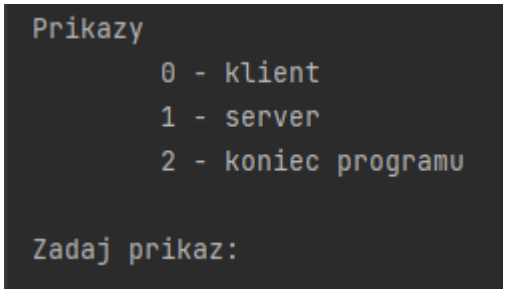
Funkcia ka_func je funkcia na vykonávanie keep-alive správ. Táto funkcia je vykonávana v samostatnom threade (vlákne). Funkcia je spustená z funkcie klient, keď sa neodosielajú dáta. A funkcia odosiela každých 5 s keep alive správu a čaká na odpoveď. Ak nedostane 3x za sebou odpoveď, tak pošle signál do funkcie client a tá vypíše, že server 3x neodpovedal na KA správu a ukončí spojenie.

```
def ka_func(active, ip_port, sockt):
    i = 0
    while not active.isSet():
        e2 = active.wait(5)
        if not e2:
            send_ka_message(ip_port[0], ip_port[1], sockt)
            sockt.settimeout(1)
            try:
                data, app_port = sockt.recvfrom(2048)
            except (socket.timeout, ConnectionResetError):
                sockt.settimeout(None)
                i += 1
                if i == 3:
                    print(
                        "\nServer neodpovedal 3 krat na Keep alive spravu..
                    send_fin_message(ip_port[0], ip_port[1], sockt)
                    sockt.close()
                    active.set()
                continue
            received = fragment_rec(data)
            if received.get_flag() != 12:
                i += 1
            if i == 3:
                print(
                    "\nServer neodpovedal 3 krat na Keep alive spravu.. Posi
                send_fin_message(ip_port[0], ip_port[1], sockt)
                sockt.close()
                active.set()
```

Obrázok 10 KA funkcia

5. POUŽÍVATEĽSKÉ PROSTREDIE

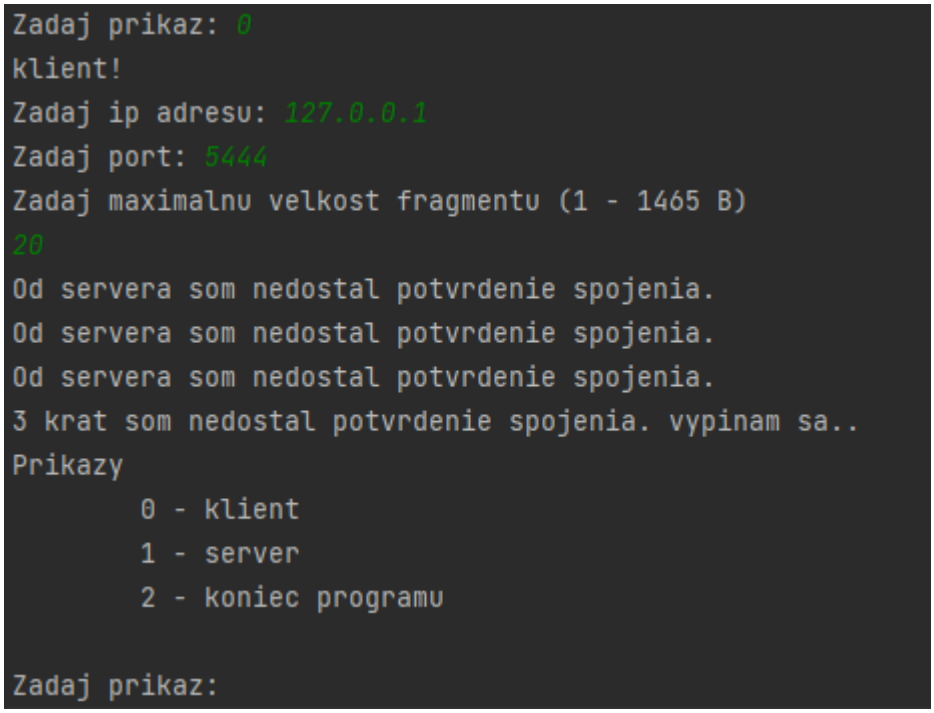
Používateľské prostredie začína v menu (obr. 11), v ktorom si používateľ zvolí server alebo klienta alebo ukončenie programu.



Obrázok 11 menu

5.1. Klient

Pri zvolení klienta (stlačením 0) sa zobrazí informácia, že je používateľ v klientovi a následne zadá používateľ zadá ip adresu servera port a maximálnu veľkosť fragmentu. Po zadaní veľkosti začne klient odosielať každé 3 sekundy inicializačné správy, ak na žiadnu z nich server neodpovie, tak sa soket zatvorí a program vráti do menu.(obr. 12)



Obrázok 12 neúspešné inicializovanie spojenia

Pri úspešnom inicializovaní spojenia so serverom sa ukáže menu, kde si môže používateľ zvoliť akú správu chce poslať M - poslanie správy, F – poslanie súboru, K - ukončenie spojenia. Pri vybraní poslania súboru musí potom používateľ zadať absolútnu cestu ku súboru, ktorý chce poslať. Po zadaní klient odošle názov súboru a čaká na ACK správu od servera. Ak do 60 s žiadna nepríde, tak spojenie sa ukončí. Ak príde ACK, tak sa začnú odosielať fragmenty. Na koniec sa vypíše počet správne odoslaných fragmentov a absolútna cesta. (obr. 13)

```
Spojenie s serverom ('127.0.0.1', 5444) prebehlo uspesne!
Typy sprav
        M - poslanie spravy
        F - poslanie suboru
        K - ukoncenie spojenia
Zadaj typ spravy:  F
Zadaj cestu k suboru ,kt. sa ide poslat:  D:\knight_tour.py
Fragment s cislom 1 s velkostou 350 bol uspesne poslany
Fragment s cislom 2 s velkostou 350 bol uspesne poslany
Fragment s cislom 3 s velkostou 350 bol chybne poslany.. posielam znova
Fragment s cislom 3 s velkostou 350 bol uspesne poslany
Fragment s cislom 4 s velkostou 350 bol uspesne poslany
Fragment s cislom 5 s velkostou 350 bol uspesne poslany
Fragment s cislom 6 s velkostou 350 bol chybne poslany.. posielam znova
Fragment s cislom 6 s velkostou 350 bol chybne poslany.. posielam znova
Fragment s cislom 6 s velkostou 350 bol uspesne poslany
Fragment s cislom 7 s velkostou 313 bol chybne poslany.. posielam znova
Fragment s cislom 7 s velkostou 313 bol uspesne poslany
Pocet spravne odoslanych fragmentov: 7
absolutna cesta k suboru: D:\knight_tour.py
Nedostal som odpoved pre zmenu.
Typy sprav
        M - poslanie spravy
        F - poslanie suboru
        K - ukoncenie spojenia
```

Obrázok 13 posielanie súboru

```
Zadaj typ spravy:  M
Zadaj spravu na poslanie:  ahoj, to som ja pato!
Fragment s cislom 1 s velkostou 2 bol uspesne poslany
Fragment s cislom 2 s velkostou 2 bol uspesne poslany
Fragment s cislom 3 s velkostou 2 bol uspesne poslany
Fragment s cislom 4 s velkostou 2 bol uspesne poslany
Fragment s cislom 5 s velkostou 2 bol uspesne poslany
Fragment s cislom 6 s velkostou 2 bol uspesne poslany
Fragment s cislom 7 s velkostou 2 bol uspesne poslany
Fragment s cislom 8 s velkostou 2 bol uspesne poslany
Fragment s cislom 9 s velkostou 2 bol uspesne poslany
Fragment s cislom 10 s velkostou 2 bol uspesne poslany
Fragment s cislom 11 s velkostou 1 bol uspesne poslany
Pocet odoslanych fragmentov 11
Nedostal som odpoved pre zmenu.
```

Obrázok 14 posielanie správy

```
Fragment s cislom 8 s velkostou 2 bol uspesne poslany
Fragment s cislom 9 s velkostou 2 bol uspesne poslany
Fragment s cislom 10 s velkostou 2 bol uspesne poslany
Fragment s cislom 11 s velkostou 1 bol uspesne poslany
Pocet odoslanych fragmentov 11
Nedostal som odpoved pre zmenu.
Typy sprav
        M - poslanie spravy
        F - poslanie suboru
        K - ukoncenie spojenia
Zadaj typ spravy:
Server neodpovedal 3 krat na Keep alive spravu.. Posielam spravu o ukonceni a vypinam sa..
 Pre pokracovanie stlac enter
```

Obrázok 15 pri neodpovedaní servera

5.2. Server

Pri zvolení servera používateľ musí zadať na ktorom porte bude server počúvať. Potom server čaká na dáta. Keď serveru príde správa s flagom I a F (správa o inicializicií posielania súboru), server si musí zvoliť, kde uložiť súbor (absolútnu cestu adresára, kde sa má súbor uložiť). Následne sa prijíma súbor.

```
Prikazy
                0 - klient
                1 - server
                2 - koniec programu

Zadaj prikaz: 1
server!
Zadaj port na, kt. mam pocuvat: 5444
Pokusa sa pripojit ('127.0.0.1', 58000).. posielam potvrdenie...
zadaj cestu kde ma byt subor ulozeny: D:
fragment cislo 1 dlzky 350 je chybny
fragment cislo 1 dlzky 350 je chybny
fragment cislo 1 dlzky 350 bol prijaty
fragment cislo 2 dlzky 350 bol prijaty
fragment cislo 3 dlzky 350 bol prijaty
fragment cislo 4 dlzky 350 bol prijaty
fragment cislo 5 dlzky 350 bol prijaty
fragment cislo 6 dlzky 350 bol prijaty
fragment cislo 7 dlzky 313 bol prijaty
Prijaty subor: D:\knight_tour.py
Pocet spravne prijatych fragmentov: 7

AK sa chces zmenit na klienta klikni ctrl+c do 3 sekund
Nestihol si pokracujeme...
```

Obrázok 16 ukážka prijímania na serveri

5.3. Výmena servera s klientom

Výmena servera s klientom prebieha po odosielaní správy alebo súboru. Vtedy si server môže vymeniť úlohy s klientom. Má na to 3 sekundy po výmene správy alebo súboru. Počas tých troch sekúnd musí server stlačiť ctrl + c. Ak stlačí, tak sa úlohy vymenia server bude klientom a klient serverom. V prípade, že nestlačí, tak sa vypíše nestihol si pokračujeme.

```
fragment cislo 19 dlzky 2 bol prijaty
fragment cislo 20 dlzky 2 bol prijaty
fragment cislo 21 dlzky 2 je chybny
fragment cislo 21 dlzky 2 bol prijaty
Prijata sprava: ahojky som tu doma a pisem dokumentaciu :)

Pocet spravne prijatych fragmentov: 21

AK sa chces zmenit na klienta klikni ctrl+c do 3 sekund
Prikazy
                0 - klient
                1 - server
                2 - koniec programu

Zadaj prikaz: klient!
Zadaj ip adresu:
```

Obrázok 17 stlačenie ctrl+c

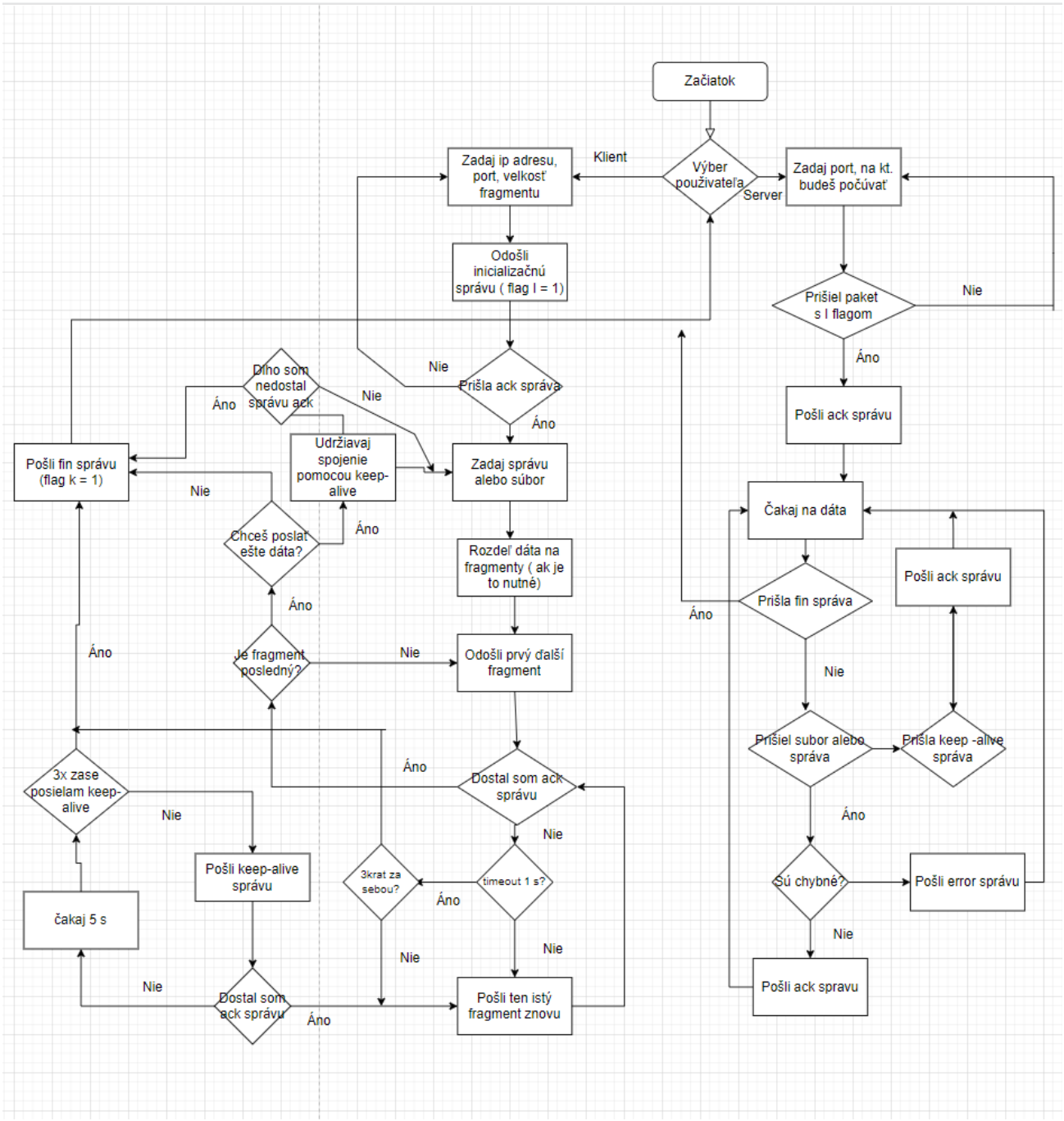
```
Fragment s cislom 21 s velkostou 2 bol chybn
Fragment s cislom 21 s velkostou 2 bol uspes
Pocet odoslanych fragmentov 21
Server chce zmenit ulohy..
Prikazy
        0 - klient
        1 - server
        2 - koniec programu

Zadaj prikaz: server!
Zadaj port na, kt. mam pocuvat:
```

Obrázok 18 klient sa zmeni na server

Výmena môže byť vykonaná aj tak, že klient ukončí spojenie a potom si v menu prehodia úlohy. Táto druhý typ metódy je na inicializovaný na strane klienta.

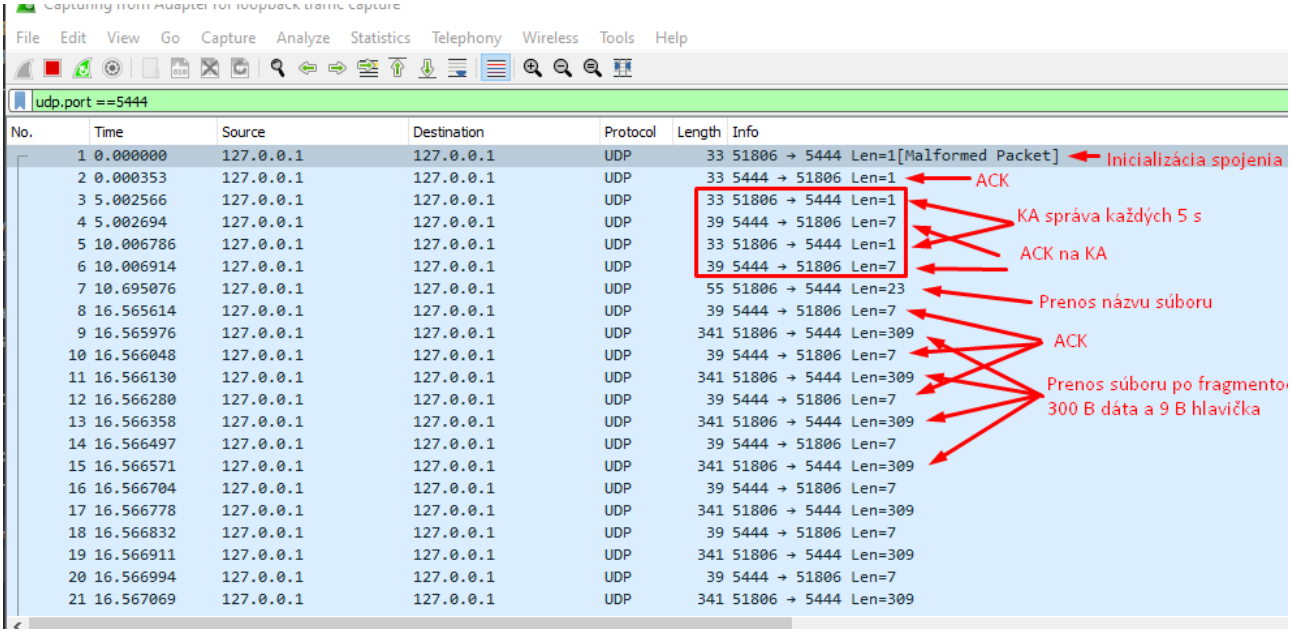
6. DIAGRAM SPRACOVÁVANIA KOMUNIKÁCIE



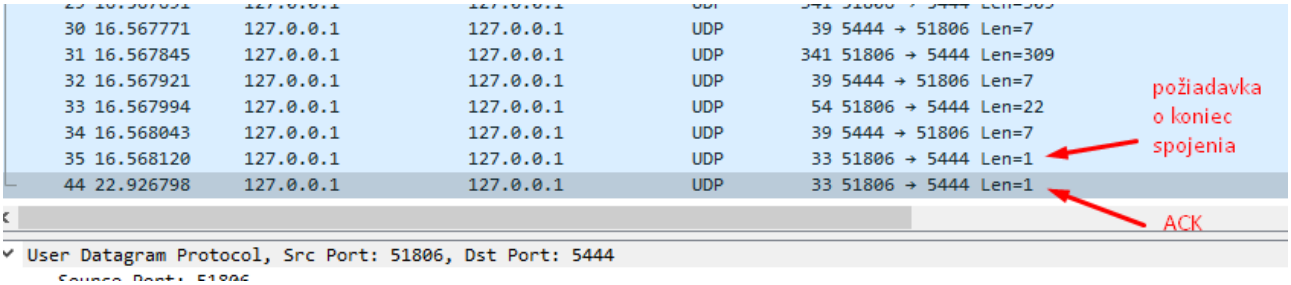
Obrázok 19 finálny diagram

7. ZOBRAZENIE VO WIRESHARK

Komunikácia je otestovaná na loopbacku. Pre zachytávanie paketov bol použitý program wireshark. Pre filtráciu tejto komunikácie je treba do filtra zadať udp.port == číslo portu.



Obrázok 20 inicializácia a prenos vo Wiresharku



Obrázok 21 koniec spojenia

8. ZÁVER

Program bol úspešne implementovaný v pythone. Program bol testovaný na jednom počítači, kde jedna inštancia programu bola spustená v pyCharm a druhá v príkazovom riadku. Otestované boli súbory rôznej veľkosti až do desiatok MB a správy rozličného druhu. Simulácia chyby bola nastavená na 20 percentnú pravdepodobnosť. Server môže inicializovať výmenu úloh ako aj klient. Bola taktiež implementovaná stop & wait ARQ metóda, ktorá čaká vždy na ACK.