

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Počítačové a komunikačné siete
Zadanie č. 2
UDP komunikátor

Autor: Marek Smutný

Sk. Rok: ZS 2022/2023

Cvičenie: štvrtok 16:00, Ing. Kristián Košťál, PhD.

Zadanie

Navrhните 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 znovuvyžiadanie chybných fragmentov, vrátane pozitívneho aj negatívneho potvrdenia. Po zapnutí programu, komunikátor automaticky odosiela paket pre udržanie spojenia každých 5s pokiaľ používateľ neukončí spojenie ručne. Odporúčame riešiť cez vlastne definované signalizačné správy a samostatný thread.

Návrh Protokolu

Typy správ využívané vysielateľom:

Štruktúra hlavičky pre 1, 2, 3:

Typ správy

Typ správy (1B)

1. Inicializácia spojenia (0000 0001 bin)
2. Keep Alive (0000 0010 bin)
3. Zmena vysielania (0000 0011 bin)

Štruktúra hlavičky pre 4, 5:

Typ správy	Počet fragmentov	Prípona súboru
------------	------------------	----------------

Typ správy (1B), Počet fragmentov (2B)

4. Informačný paket text – vysielateľ odošle predtým ako začne posielať dátové pakety s textom (0000 0100 bin)
5. Informačný paket súbor – vysielateľ odošle predtým ako začne posielať dátové pakety so súborom (0000 0101 bin)

Štruktúra hlavičky pre 6, 7:

Typ správy	Číslo fragmentu	CRC	Dáta
------------	-----------------	-----	------

Typ správy (1B), Číslo fragmentu (2B), CRC(4B), Dáta = 1500B – 8B – 20B – 7B = 1465B (1500B je maximálna veľkosť dát, ktorú je možné poslať bez fragmentovania na linkovej vrstve)

6. Dátový paket text (0000 0110 bin)
7. Dátový paket súbor (0000 0111 bin)

Typy správ využívané prijímačom:

Štruktúra hlavičky pre ACK a ERR:

Typ správy	Číslo fragmentu
------------	-----------------

Typ správy (1B), Číslo fragmentu (2B)

1. ACK - potvrdenie spojenia, doručenia fragmentu, zmeny vysielania (0000 0001 bin)
2. ERR – dáta prišli poškodené (0000 0010 bin)

ARQ metóda

Pre moje riešenie som si vybral Stop and wait ARQ metódu. Táto metóda funguje tak, že vysielateľ pošle packet s dátami a potom čaká na potvrdenie od prijímača. V prípade, že príde kladná odozva, posiela ďalší fragment v poradí. Ak prijímačovi príde poškodený paket, pošle zápornú odozvu (error) a vysielateľ následne znovu odošle posledný odoslaný fragment. Nevýhoda tejto metódy je jej rýchlosť, vysielateľ musí čakať na odozvu a až potom môže odoslať ďalší fragment.

Metóda na udržanie spojenia

Po nadviazaní spojenia bude vysielateľ každých 5 sekúnd posielať Keep Alive paket. Taktiež po nadviazaní spojenia sa na prijímačovi spustí timer, ktorý ak každých 30 sekúnd nedostane Keep Alive paket, spojenie preruší. Taktiež ak vysielateľ nedostane potvrdenie po odoslaní Keep Alive paketu, prijímač pravdepodobne ukončil spojenie a taktiež ukončí spojenie.

Metóda kontrolnej sumy CRC

Na výpočet checksum budem používať metódu `crc32()` z knižnice `zlib`. Táto funkcia využíva 32 bitový polynóm: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

Tento polynóm vyzerá v binárnom formáte takto: 1 0000 0100 1100 0001 0001 1101 1011 0111

Algoritmus na vypočítanie crc:

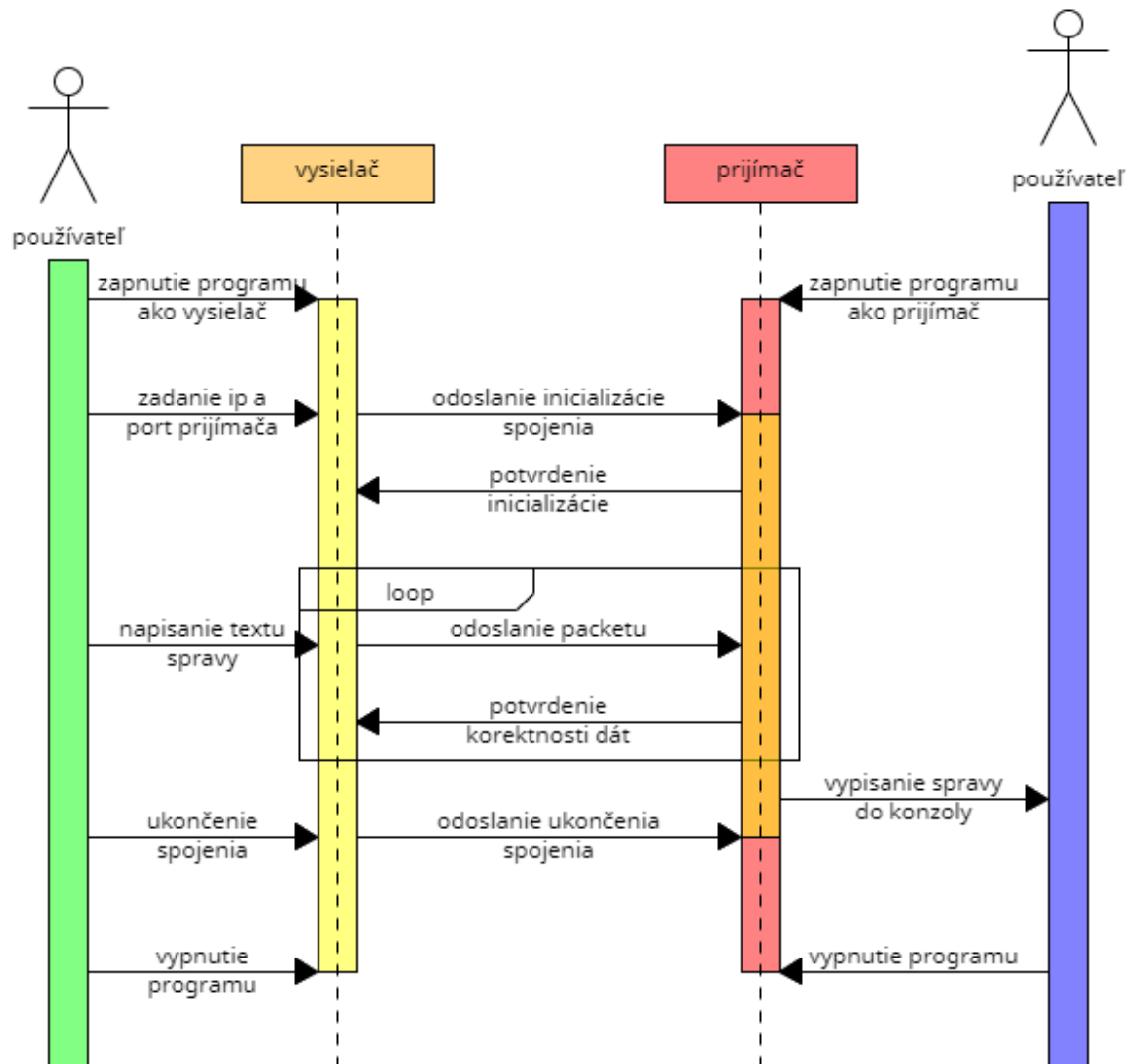
1. Obráť vstup
2. Na koniec pridaj 32 núl
3. Urob XOR s 0xFFFFFFFF = 1111 1111 1111 1111 1111 1111 1111 1111
4. Urob XOR s polynómom ak je na začiatku 1, ak nie posuň medzivýsledok doľava
5. Opakuj krok 4. kým prvých x bitov nie je 0, x = počet bitov vstupu na začiatku
6. Urob XOR s 0xFFFFFFFF
8. Obráť výsledok

Výpočet crc písmena „a“.

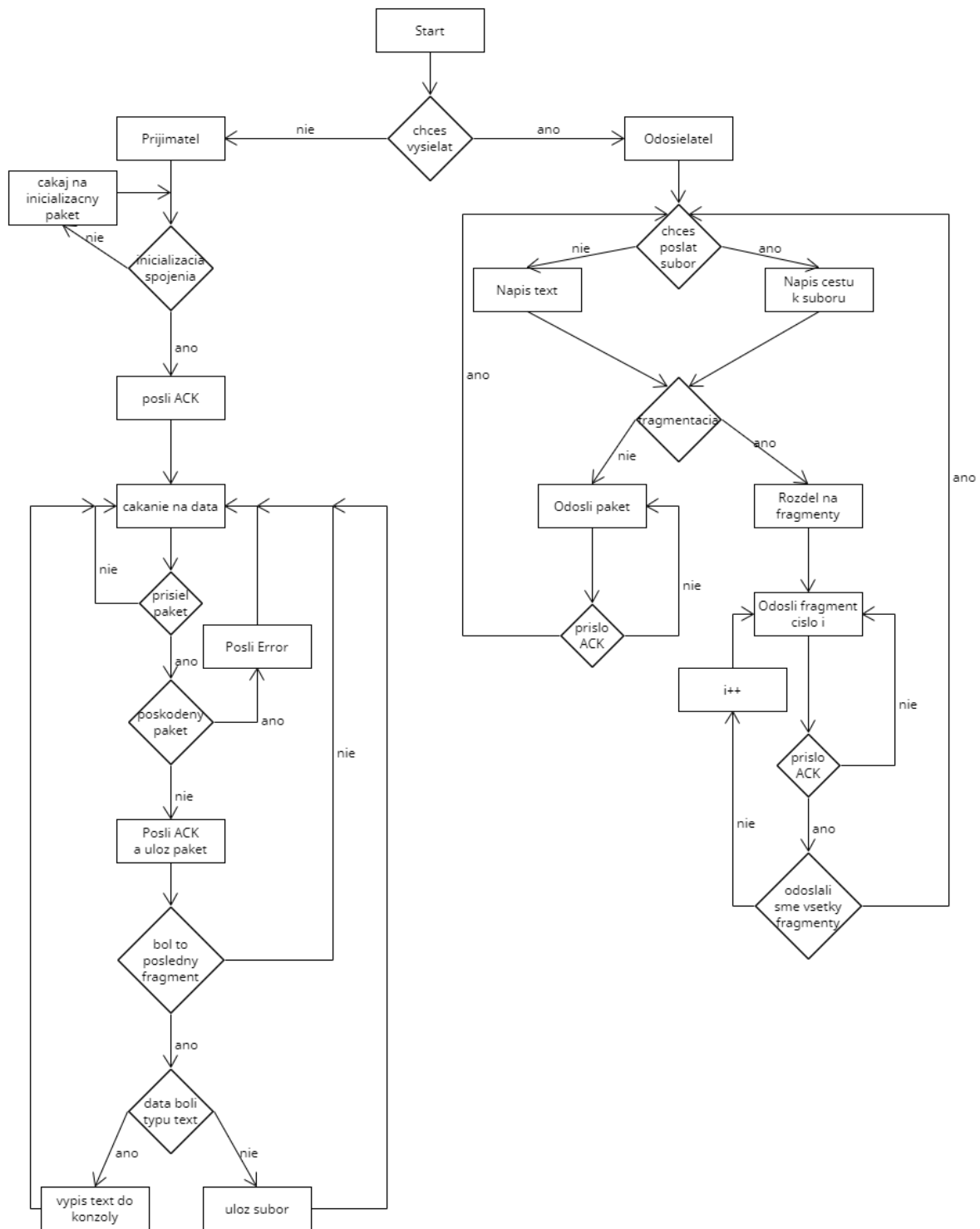
[illegible]

Diagram spracovávania komunikácie

Sekvenčný diagram:



Flowchart:



Zmeny v porovnaní s návrhom

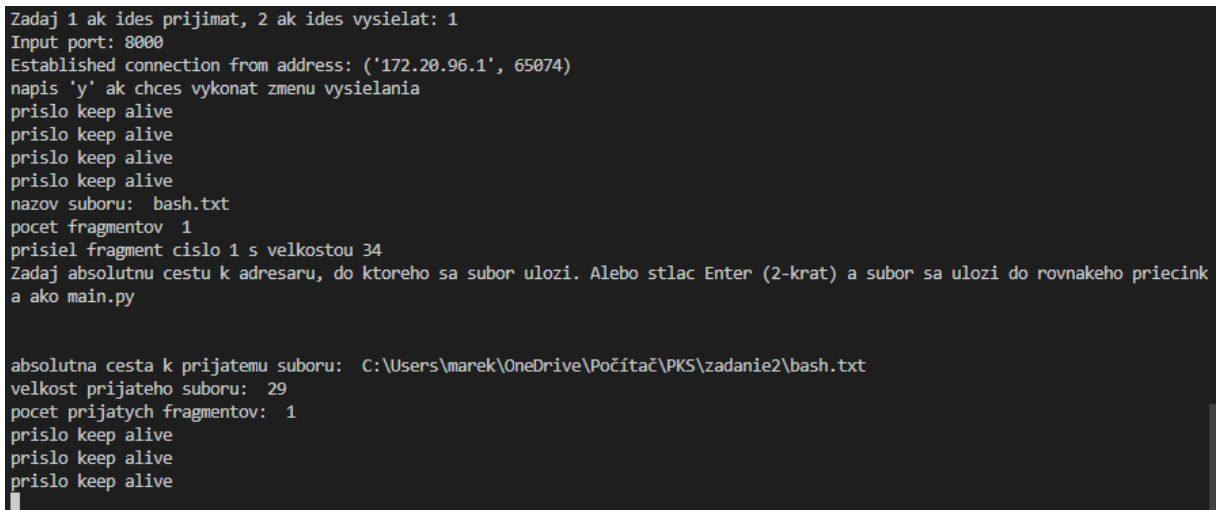
Po konzultácii s cvičiacim, som vyskúšal viaceré funkcie crc a počas toho, som zistil, že crc32 z knižnice zlib nemusí byť najlepšia voľba. Nakoniec som vo svojom riešení použil funkciu crc_hqx z knižnice binascii. Táto funkcia využíva CRC-CCITT polynóm (10001000000100001), čo znamená, že v porovnaní so zlib.crc32 je výsledný zvyšok menší, konkrétne 2 bajty. Tým pádom sa uvoľnia 2 bajty, ktoré môžeme využiť na prenos dát. Toto bol jeden z dôvodov, že som si zvolil práve crc_hqx z knižnice binascii.

Taktiež v porovnaní s návrhom pribudla funkcionálna zmena vysielania zo strany prijímača. V prípade, že používateľ na strane prijímača sa rozhodne pre zmenu roly (swap) obidve strany si ju vymenia. Podrobnejšie v časti Dôležité časti kódu.

Užívateľské rozhranie

Používateľ a program komunikujú výlučne pomocou konzoly. Program vypisuje inštrukcie na správne ovládanie programu a čaká na vstup od používateľa.

Obrázok 1



```
Zadaj 1 ak ides prijimat, 2 ak ides vysielat: 1
Input port: 8000
Established connection from address: ('172.20.96.1', 65074)
napis 'y' ak chces vykonat zmenu vysielania
prislo keep alive
prislo keep alive
prislo keep alive
prislo keep alive
nazov suboru: bash.txt
pocet fragmentov 1
prisel fragment cislo 1 s velkostou 34
Zadaj absolutnu cestu k adresaru, do ktoreho sa subor ulozi. Alebo stlac Enter (2-krat) a subor sa ulozi do rovnakeho priecku
a ako main.py

absolutna cesta k prijatemu suboru: C:\Users\marek\OneDrive\Počítač\PKS\zadanie2\bash.txt
velkost prijateho suboru: 29
pocet prijatych fragmentov: 1
prislo keep alive
prislo keep alive
prislo keep alive
```

Na obrázku vyššie vidíme, že používateľ si vybral rolu prijímača, nastavil port, nadviazalo sa spojenie a prišiel mu súbor bash.txt, ktorý uložil do priečinka k main.py.

Obrázok 2

```
Zadaj 1 ak ides prijimat, 2 ak ides vysielat: 2
Input IP address of server (localhost): 172.20.96.1
Input port: 8000
Connected to address: ('172.20.96.1', 8000)
Zadaj 4 ak chces poslat text, 5 ak chces poslat subor, 3 ak chces zmenit vysielanie: 5
napis nazov suboru: bash.txt
zadaj max. velkost fragmentu (max 1463): 1463
absolutna cesta k vybranemu suboru: C:\Users\marek\OneDrive\Počítač\PKS\zadanie2\bash.txt
velkost suboru: 29
pocet fragmentov: 1
Zadaj 4 ak chces poslat text, 5 ak chces poslat subor, 3 ak chces zmenit vysielanie: [ ]
```

Na druhom obrázku vidíme, že používateľ si zvolil rolu vysieláča, zadal ip prijímača a port. Následne si zvolil, že chce poslať súbor, napísal jeho názov a veľkosť fragmentu.

Obrázok 3

```
Zadaj 4 ak chces poslat text, 5 ak chces poslat subor, 3 ak chces zmenit vysielanie: 3

Zatvaram spojenie

Input port: 8000
Established connection from address: ('172.20.96.1', 53495)
prislo keep alive
napis 'y' ak chces vykonat zmenu vysielania
prislo keep alive
```

Na treťom obrázku vidíme, že vysieláč zmenil vysielanie, spojenie sa uzatvorilo a nakoniec zadal port, na ktorom bude počúvať ako prijímač a vytvorilo sa nové spojenie.

Obrázok 4

```
prisla zmena vysielania stlac ENTER
Input IP address of server (localhost): 172.20.96.1
Input port: 8000
Connected to address: ('172.20.96.1', 8000)
Zadaj 4 ak chces poslat text, 5 ak chces poslat subor, 3 ak chces zmenit vysielanie: [ ]
```

Na štvrtom obrázku vidíme zmenu vysielania z pohľadu prijímača. Program mu oznámil, že prišla zmena vysielania, následne zadal ip prijímača a port a vytvorilo sa spojenie.

Kontrola vo Wiresharku

Keep alive správy:

Obrázok 5

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	169.254.142.15	169.254.154.137	UDP	43	56477 → 8000 Len=1
2	0.002698	169.254.154.137	169.254.142.15	UDP	60	8000 → 56477 Len=1
6	5.008940	169.254.142.15	169.254.154.137	UDP	43	56477 → 8000 Len=1
7	5.011603	169.254.154.137	169.254.142.15	UDP	60	8000 → 56477 Len=1
12	10.022148	169.254.142.15	169.254.154.137	UDP	43	56477 → 8000 Len=1
13	10.025045	169.254.154.137	169.254.142.15	UDP	60	8000 → 56477 Len=1

Posielanie súboru:

Obrázok 6

103	76.090796	169.254.142.15	169.254.154.137	UDP	69 56477 → 8000 Len=27
104	76.093708	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1
105	76.093837	169.254.142.15	169.254.154.137	UDP	1500 56477 → 8000 Len=1458
106	76.098968	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1
107	76.099483	169.254.142.15	169.254.154.137	UDP	1500 56477 → 8000 Len=1458
108	76.101762	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1
109	76.102029	169.254.142.15	169.254.154.137	UDP	1500 56477 → 8000 Len=1458
110	76.104184	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1
111	76.104514	169.254.142.15	169.254.154.137	UDP	1500 56477 → 8000 Len=1458
112	76.106785	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1
113	76.106972	169.254.142.15	169.254.154.137	UDP	1500 56477 → 8000 Len=1458
114	76.109510	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1
115	76.109925	169.254.142.15	169.254.154.137	UDP	1500 56477 → 8000 Len=1458
116	76.114724	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1

Na obrázku 6 môžeme vidieť začiatok procesu posielania fragmentov súboru. Prvý packet je informácia pre prijímač, že mu príde súbor. Následne sa posielajú všetky fragmenty súboru.

Obrázok 7

533	76.574885	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1
534	76.575054	169.254.142.15	169.254.154.137	UDP	1500 56477 → 8000 Len=1458
535	76.576385	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1
536	76.576509	169.254.142.15	169.254.154.137	UDP	1500 56477 → 8000 Len=1458
537	76.577898	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1
538	76.577990	169.254.142.15	169.254.154.137	UDP	991 56477 → 8000 Len=949
539	76.579382	169.254.154.137	169.254.142.15	UDP	60 8000 → 56477 Len=1
551	80.161027	169.254.142.15	169.254.154.137	UDP	43 56477 → 8000 Len=1

Na obrázku 7 vidíme posledné fragmenty súboru (posledný má 991 bajtov) a jednu správu keep alive.

Dôležité časti kódu

Fragmentácia súboru a odosielanie fragmentov:

Obrázok 8

```
#poslanie suboru
elif urob == "5":
    name = input("napis nazov suboru: ")
    fragment_size = int(input("zadaj max. velkost fragmentu (max 1453): "))

    #otvorenie suboru a vypisanie absolutnej cesty a
    file = open(name, "rb")
    file_size = os.path.getsize(name)
    print("absolutna cesta k vybranemu suboru: ", os.path.abspath(name))
    print("velkost suboru: ", file_size)

    #vypocet poctu fragmentov
    pocet_fragmentov = math.ceil(file_size/fragment_size)
    print("pocet fragmentov: ", pocet_fragmentov)
    sprava = file.read()

    #poslanie informacneho packetu
    vysielac_socket.sendto(struct.pack("c", str.encode("5")) + struct.pack("H", pocet_fragmentov) + str.encode(os.path.basename(name)), prijimac_address)
    vysielac_socket.settimeout(30)
    data, address = vysielac_socket.recvfrom(1500)
    data = data.decode()

    if data == "1":
        cislo_fragmentu = 1

        #fragmentacia suboru
        while len(sprava) != 0:
            fragmentovana_sprava = sprava[:fragment_size]

            #vytvorenie hlavicky bez crc a vypocet crc
            header = struct.pack("c", str.encode("7")) + struct.pack("H", cislo_fragmentu)
            crc = binascii.crc_hqx(header + fragmentovana_sprava, 0)

            #vnesenie chyby
            if random.randint(0, 100) == 1:
                crc += 1

            #vytvorenie hlavicky s crc a poslanie packetu
            header = struct.pack("c", str.encode("7")) + struct.pack("HH", cislo_fragmentu, crc)
            vysielac_socket.sendto(header + fragmentovana_sprava, prijimac_address)
            data, address = vysielac_socket.recvfrom(1500)
            data = data.decode()

            #ak pride ack, zvys cislo fragmentu a zahod poslane data
            if data == "1":
                cislo_fragmentu += 1
                sprava = sprava[fragment_size:]
```

Táto časť kódu (obrázok 8) sa nachádza vo funkcii vysielac. V prípade, že používateľ na strane vysieláča zvolí možnosť odoslať súbor, program si vypýta názov súboru a maximálnu veľkosť fragmentu. Do názvu súboru je možné zadať aj absolútnu cestu k súboru alebo v prípade, že používateľ zadá iba názov, súbor sa musí nachádzať v rovnakom priečinku ako kód programu. Následne program vypočíta počet fragmentov a odošle informačný packet s počtom fragmentov a názvom súboru. Ak od prijímača príde potvrdenie, fragmentuje prečítaný súbor, počíta crc, vnáša chyby a odosiela fragmenty vždy, keď dostane potvrdenie od prijímača o predošlom fragmente. Šanca, že program vnesie chybu je 1:101 a v podstate je to zmena crc hodnoty.

Keep alive:

Obrázok 9

```
#vytvorenie threadu pre keep alive
thread = threading.Thread(target=keep_alive, args=(vysielac_socket, prijimac_address, 5))
thread.start()
```

Vytvorenie threadu pre keep alive (obrázok 9) sa nachádza vo funkcii vysielac_login. Tento thread sa vytvorí po úspešnom nadviazaní spojenia. Funkcia keep_alive (obrázok 10) každých 5 sekúnd posieľa keep alive packet a kontroluje, či prijímač náhodou nevyžiadal zmenu role (swap).

Obrázky 11

11

Táto časť kódu (obrázok 11) sa nachádza vo funkcii `prijimac`. V prípade, že prijímač dostane packet typu 5 (informačný packet súbor) vypíše názov súboru a počet fragmentov a odošle potvrdenie. Následne v loope prijíma fragmenty súboru a keep alive správy. Po prijatí fragmentu skontroluje crc a v prípade, že je správne, vypíše číslo fragmentu a jeho veľkosť nakoniec uloží dáta do listu `cela_sprava`. Potom ako prijme posledný fragment, používateľ môže zadať absolútnu cestu adresára určeného na uloženie súboru alebo nemusí zadať nič a súbor sa uloží do adresára, v ktorom sa nachádza kód programu. Nakoniec vypíše absolútnu cestu k uloženému súboru, jeho veľkosť a počet prijatých fragmentov.

Výmena role na strane prijímača:

Obrázok 12

```
#funkcia prijimaca beziaca sa samostatnom threade pocuva swap
def listen_swap():
    global zmena
    swap = input("napis 'y' ak chces vykonat zmenu vysielania\n")
    if swap == 'y':
        zmena = True

#funkcia, ktora vytvara alebo zatvara thread
def swap_thread(on):
    global thread1
    if on and thread1 == None:
        thread1 = threading.Thread(target=listen_swap, args=())
        thread1.start()
    elif not on:
        thread1.join()
        thread1 = None
```

Funkcia `swap_thread` (obrázok 12) zapína a vypína thread, na ktorom beží funkcia `listen_swap`. Táto funkcia obsahuje input. Keď používateľ na strane prijímača napíše „y“, prijímač nebude na keep alive správy odpovedať potvrdením „1“ ale „3“, čo spôsobí, že vysielateľ taktiež zmení rolu. Zmena roly je možná aj na strane vysielateľa ale toto riešenie je jednoduché a nezaslúži si dostať sa do dokumentácie. 😊