Komunikácia s využitím UDP protokolu

Zadanie 2

Skrypova Karolina ID: 123423

Zadanie úlohy

Navrhnite a implementujte program spoužitím 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 zdvoch častí – vysielacej a prijímacej. Vysielací uzol pošle súbor inému uzlu v sieti. Predpokladá sa, že vsieti dochádza kstratám 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 oprijatí fragmentu s jeho poradím ači bol prenesený bez chýb. Po prijatí celého súboru na cieľovom uzle tento zobrazí správu ojeho prijatí aabsolútnu 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 samostatné vlákno.

Skrypova Karolina ID: 123423

Návrh protokolu:

Type	Fragment size	Size of message	Number of fragments	Packet order	Crc	Data

Type (1 byte):

1. 0000 0001 – initializacia spojenia

2. 0000 0010 – keep alive

3. 0000 0011 – vymenit server a klienta

4. 0000 0100 – prenos spravy

5. 0000 0101 – prenos suboru

6. 0000 0110 – skoncit spojenie

8. 0000 1000 – zaciatok prenosu súborov, prenos názvu súboru

Type (1 byte) for server

5. 0000 0101 – fragment of text or filewith rigth ack was received(ACK)

7. 0000 0111 - fragment with wrong crc was received(NACK)

Fragment size(2 bytes):

Používateľ má možnosť vybrať veľkosť fragmentu manuálne pred každým odoslaním správy alebo súboru.

Počet fragmentov(2 bytes):

Počet fragmentov sa vypočíta podľa nasledujúceho vzorca a tiež sa zaokrúhli na väčšie číslo

$$Num\ of\ fragments = \frac{size\ of\ message}{fragment\ size}$$

Crc(2 bytes):

použitie funkcie crc_hqx z modulu binascii v jazyku Python na výpočet 16-bitovej hodnoty CRC (Cyclic Redundancy Check) pre údaje v premennej part_message

$$crc\ =\ binascii.crc\_hqx(part\_message, 0)$$

Na výpočet checksum použijem crc16 z knižnice binascii. $\boldsymbol{x\ 16 + x\ 12 + x\ 5 + 1}$

Binarny format: 1000100000010001

Cyklická kontrola redundancie (CRC) je typ kódu na kontrolu chýb, ktorý sa široko používa v komunikačných systémoch na zisťovanie chýb v prenášaných údajoch. Výber konkrétneho

variantu CRC, ako je CRC-16, závisí od požiadaviek aplikácie a vlastností prenášaných údajov. Tu je niekoľko dôvodov, prečo som vybrala prave CRC-16:

1. Rovnováha medzi výkonom a veľkosťou: CRC-16 dosahuje rovnováhu medzi schopnosťou detekcie chýb a veľkosťou kontrolného súčtu. Poskytuje primeranú úroveň detekcie chýb pri použití 16- bitového kontrolného súčtu, vďaka čomu je efektívnejší z hľadiska ukladania a spracovania v porovnaní s dlhšími variantmi CRC.

2. Vhodnosť pre krátke dátové rámce: V scenároch, kde sú dátové rámce relatívne krátke, je výhodné použiť kratší CRC ako CRC-16. Dlhšie varianty CRC môžu byť pre krátke správy prehnané, čo vedie k zbytočnej výpočtovej réžii.

ARQ:

Pre moje riešenie som si vybrala Stop and wait ARQ metódu. Stop-and-Wait ARQ (Automatic Repeat ReQuest) je jednoduchý protokol kontroly chýb v dátovej komunikácii. Odosielateľ odošle jeden rámec a čaká na potvrdenie (ACK) od príjemcu. Ak je prijaté ACK, odosielateľ odošle nasledujúci rámec; v opačnom prípade znova odošle rovnaký rámec. Tento proces zaisťuje spoľahlivý prenos údajov, ale môže byť neefektívny, najmä v prostrediach s vysokou latenciou alebo v prostrediach náchylných na chyby. Nevýhodou tejto metódy je, že používateľ musí čakať. Stále uvažujem o zmene metódy pri následnej implementácii.

Vysvetlenie fungovania programu:

Na začiatku programu si používateľ môže vybrať, či chce byť server alebo klient, ak si vyberie, že chce byť klient, potom zadá IP adresu a port servera a pokúsi sa pripojiť, ak sa mu po 5 sekundách nepodarí pripojiť, napíše sa chyba a klient sa pokúsi znova zadať IP adresu a port, kým sa nepripojí k serveru.

Ak si používateľ vybral server, zadá jeho port a bude čakať, kým sa k nemu používateľ pripojí 30 sekúnd, ak sa nikto nepripojí, server napíše, že čas čakania klienta vypršal a dostane možnosť vybrať nový port.

Ďalej je klient:

Klient si môže vybrať 3 možnosti: 1 - odpojiť sa od servera, 2 - vymeniť si miesto so serverom, 3 - poslať správu serveru.

Ak si klient vyberie 1, pošle serveru správu typu 6(0000 0110), čo znamená, že chce ukončiť komunikáciu, po tom, ako klient dostane potvrdenie od servera, sa odpojí a potom si môže vybrať iný server, ku ktorému sa pripojí, alebo úplne ukončiť svoju prácu.

Ak klient zvolil možnosť 2, chce si so serverom vymeniť miesto. Najprv mu pošle správu s typom 3 a po prijatí rovnakého typu od servera uzavrie soket pre klienta, počká 3 sekundy, kým server uzavrie svoj soket, a potom zavolá funkciu, ktorá vytvorí server pre tohto klienta.

Ak si klient vyberie typ 3, spustí sa funkcia send to server, kde si klient môže vybrať, či chce poslať súbor alebo správu. Ďalej klient zaznamená veľkosť fragmentov, ktoré sa majú poslať, a potom, ak sa klient rozhodne poslať správu, zadá správu do príkazového riadka. Ak si vybral súbor, zadá názov súboru a potom sa súbor načíta.

Potom sa k niektorým fragmentom pridajú chybné srs, aby sa simulovala chyba. Kým sa údaje nevyčerpajú, fragment sa od neho oddelí a odošle na server. Po každom fragmente klient dostane informáciu o tom, či bol fragment prijatý alebo nie, konkrétne dostane v odpovedi typ 5 alebo 7. Ak klient dostane od servera odpoveď 5, znamená to, že fragment bol správny a prijatý, ak dostane 7, znamená to, že fragment obsahoval chybu a klient pošle fragment znova.

Strana servera:

Server má 2 možnosti, 1 je ukončiť alebo kliknúť na čokoľvek a pokračovať v čakaní na správu od klienta.

Ak server zvolí 1, uzavrie socket a program sa ukončí, ak zvolí niečo iné, čaká na správu od klienta a potom prejde na funkciu prijímania.

Vo funkcii receive server kontroluje typ prijatej správy od klienta, ak je typ 2, potom prišiel na udržiavanie komunikácie, a to je keep alive. Server potom odošle klientovi odpoveď s keep alive, čo znamená, že komunikácia bola úspešne udržiavaná. Ak je typ 3, znamená to,

že klient si chce vymeniť miesto so serverom, po čom server pošle klientovi potvrdenie a prepne na klienta.

Ak je typ správy 4, server uloží fragment správy do listu, ak je fragment posledný, pristúpi k vypísaniu výsledku. Ak fragment prišiel s chybou, server pošle klientovi oznámenie, aby klient fragment poslal znova. Ak bol fragment správny, server pošle klientovi oznámenie, aby klient mohol odoslať pakety ďalej.

Ak je typ správy 8, znamená to, že klient odoslal názvy súborov a odošle ďalšie údaje zo súboru. V tomto prípade server vráti klientovi správu typu 5, čo znamená, že server úspešne prijal údaje.

Ak je typ správy 5, potom server prijal fragment súboru od klienta a uloží ho do listu spolu so zodpovedajúcim číslom fragmentu, aby mohol súbor neskôr obnoviť.

Keď server prijme všetky fragmenty, zoradí list podľa slušných čísel fragmentov a potom vypíše počet fragmentov, veľkosť fragmentu a veľkosť súboru, ak sme predtým prijali textové správy, server vypíše výsledok do konzoly. Ak sme predtým prijali súbory, uloží údaje do nového súboru.

Potom bude mať server možnosť vymeniť sa s klientom, ak zvolí áno, pošle správu klientovi a odpojí server a potom sa vymení s klientom.

Funkcia na udržiavanie komunikácie

Aby sme mohli udržiavať komunikáciu, vytvorili sme druhé vlákno, v ktorom keep alive každých 5 sekúnd pošle serveru správu typu 2, ak klient nedostane odpoveď do 30 sekúnd, znamená to, že server už nie je aktívny a komunikácia sa ukončí. Keď klient odošle správu na server, kep alive sa preruší pripojením k hlavnému vláknu. Po úspešnom prijatí správy opäť spustíme keep alive na podporu komunikácie.

Zmieny, ktoré nastali v implementácii oproti návrhu

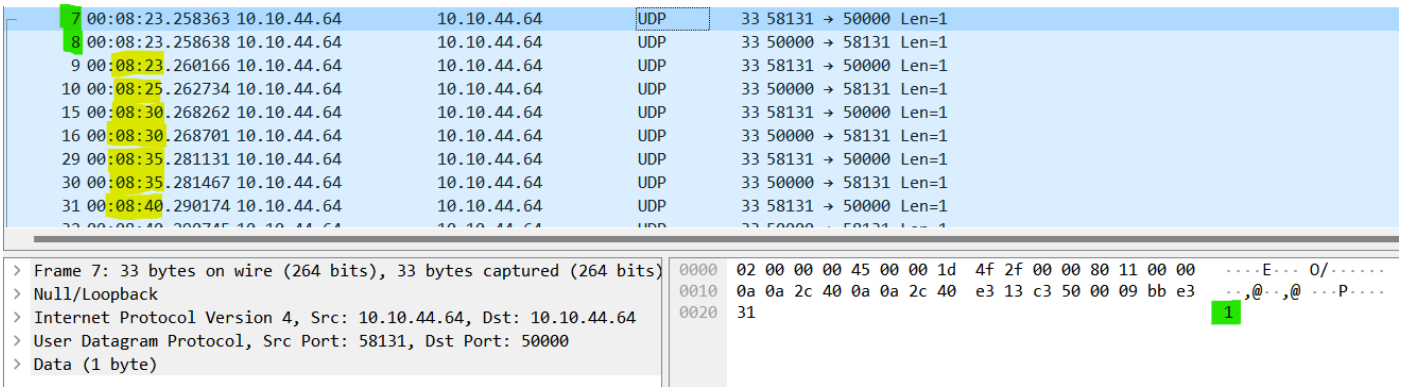
Počas implementácie programu som do hlavičky protokolu pridala veľkosť odoslanej správy, aby bolo možné vypočítať veľkosť odoslaného súboru. Boli pridané aj ďalšie typy, teraz bude pri odosielaní textu typ 4 a pri odosielaní súboru typ 5 a pridala som nový typ 8, ktorý bude zodpovedný za odosielanie názvu súboru. Tiež som pridala typ 6, ktorý bude ukončovať komunikáciu.

Príklad v programe wireshark

Po pripojení klienta k serveru odošle serveru správu typu 1 a po odpovedi servera s rovnakou správou sa na termináli zobrazí úspešné pripojenie.

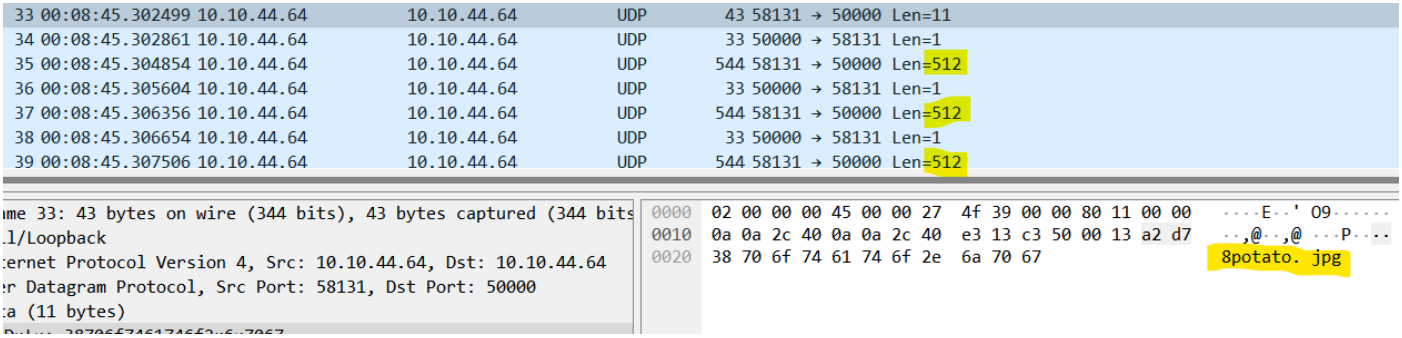
```
Choose client or server
1. Client
2. Server
Your choice: 2
Input port: 50000
Connected to client with address:  ('10.10.44.64', 62484)
```

V programe weirshark môžeme vidieť úvodné 2 inicializačné správy od klienta k serveru a od servera k klientu.



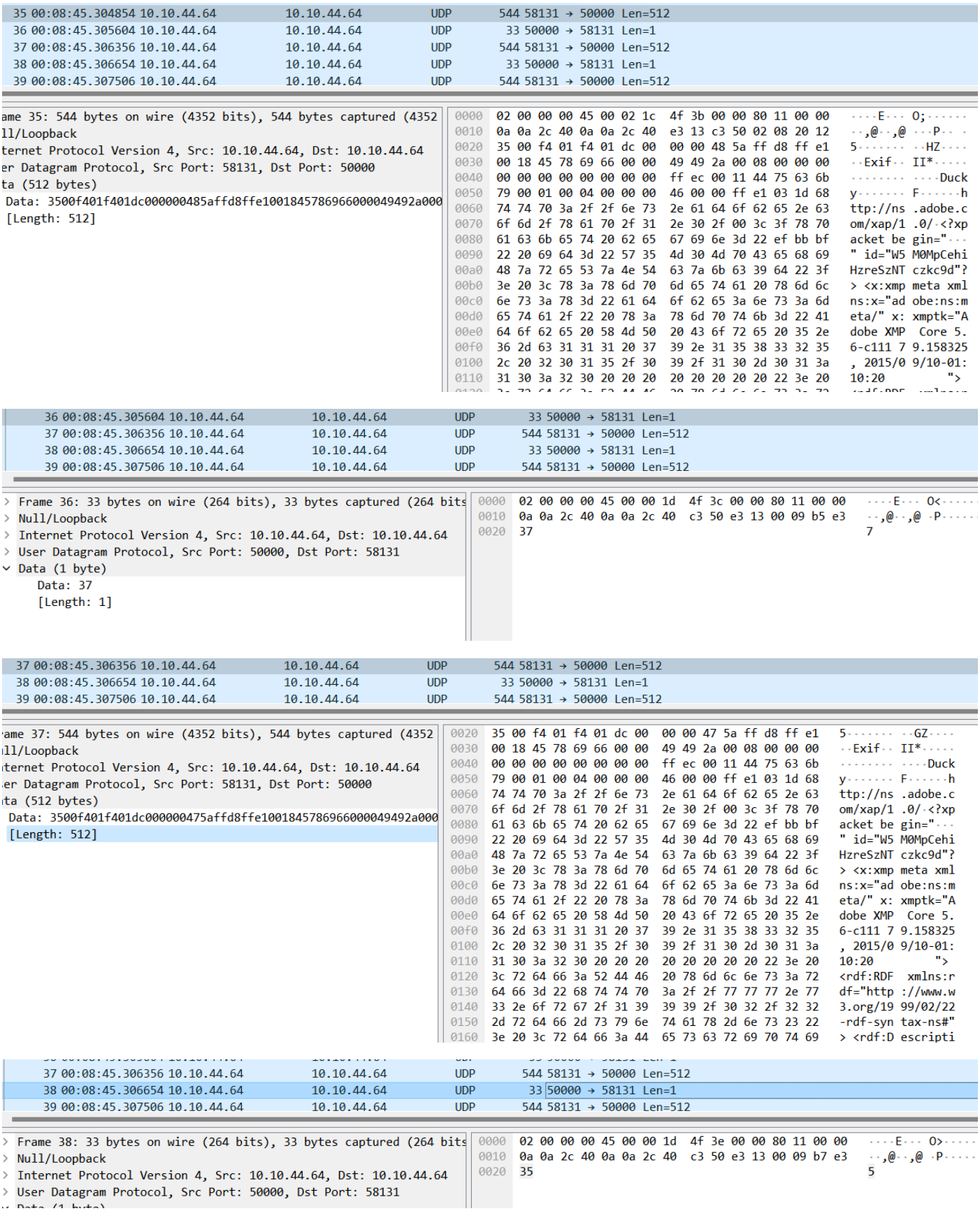
Potom má server možnosť ukončiť spojenie alebo začať čakať na správy od klienta, ak si vyberie možnosť 2, potom si server a klient budú každých 5 sekúnd vymieňať správy typu 2. Ako vidíme na nasledujúcom obrázku, správy keep alive sa posielajú každých 5 sekúnd na server a potom dostanú odpoveď.

Ďalej odošleme správu s názvom súboru pre ďalší prenos súboru.



Ako vidíte na obrázku, dostali sme správu s typom 8 a názvom súboru, potom dostaneme potvrdenie od servera a potom začneme posielať fragmenty rovnakej veľkosti 500 + 12 (veľkosť hlavičky), po každom fragmente server pošle potvrdenie o prijatí súboru.

Prvý fragment bol chybný, preto server vrátil NACK a klient poslal rovnaký fragment, po ktorom ho server potvrdil.



Ako vidíte na obrázku, dvakrát sme poslali ten istý paket a prvýkrát sme dostali odpoveď 7 a druhýkrát sme dostali odpoveď 5, čo znamená, že paket bol prijatý.

Skrypova Karolina ID: 123423

472	00:08:45.372305	10.10.44.64	10.10.44.64	UDP	33	50000 → 58131 Len=1
473	00:08:45.372446	10.10.44.64	10.10.44.64	UDP	544	58131 → 50000 Len=512
474	00:08:45.372483	10.10.44.64	10.10.44.64	UDP	33	50000 → 58131 Len=1
475	00:08:45.372560	10.10.44.64	10.10.44.64	UDP	269	58131 → 50000 Len=237
476	00:08:45.372591	10.10.44.64	10.10.44.64	UDP	33	50000 → 58131 Len=1
477	00:09:01.764432	10.10.44.64	10.10.44.64	UDP	33	50000 → 58131 Len=1

Posledný fragment bol menší ako všetky ostatné, čo znamená, že sme súbor úspešne preniesli.

Na predchádzajúcich obrázkoch bol príklad komunikácie a odosielania súboru na jednom počítači, ďalší príklad sa bude týkať možnosti výmeny klienta a servera, ako aj odosielania textu alebo súboru na rôznych počítačoch.

Najprv sa inicializuje spojenie pomocou správ typu 1, potom si klient a server vymieňajú keep alive, kým sa nezačne odosielať súbor. Je dôležité poznamenať, že keep alive sa posiela aj vtedy, keď používateľ zadáva správu alebo názov súboru.

56	12:01:44.122326	172.20.10.6	172.20.10.3	UDP	43	55921 → 50000 Len=1
57	12:01:44.123070	172.20.10.3	172.20.10.6	UDP	43	50000 → 55921 Len=1
58	12:01:44.128358	172.20.10.6	172.20.10.3	UDP	43	55921 → 50000 Len=1
64	12:01:46.316393	172.20.10.3	172.20.10.6	UDP	43	50000 → 55921 Len=1
92	12:01:51.391742	172.20.10.6	172.20.10.3	UDP	43	55921 → 50000 Len=1
93	12:01:51.392323	172.20.10.3	172.20.10.6	UDP	43	50000 → 55921 Len=1
116	12:01:56.400557	172.20.10.6	172.20.10.3	UDP	43	55921 → 50000 Len=1
117	12:01:56.400965	172.20.10.3	172.20.10.6	UDP	43	50000 → 55921 Len=1
138	12:02:01.416461	172.20.10.6	172.20.10.3	UDP	43	55921 → 50000 Len=1
139	12:02:01.417045	172.20.10.3	172.20.10.6	UDP	43	50000 → 55921 Len=1
161	12:02:06.438173	172.20.10.6	172.20.10.3	UDP	59	55921 → 50000 Len=17
162	12:02:06.438710	172.20.10.3	172.20.10.6	UDP	43	50000 → 55921 Len=1

Pri testovaní bol prvý fragment odoslanej správy vždy nesprávny a bol odoslaný druhýkrát.

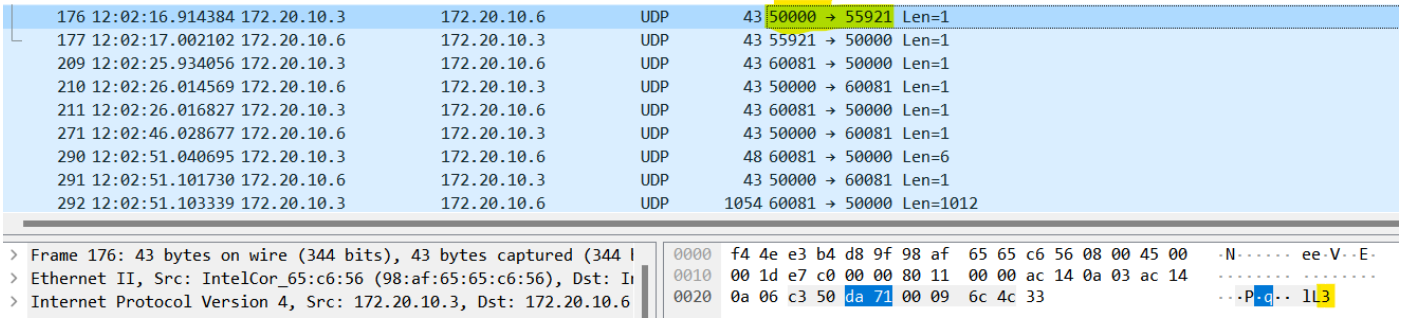
Terminál vypíše číslo fragmentu, v ktorom sa vyskytla chyba, tu bola chyba v prvom fragmente, ale keďže v programovaní počítame od nuly, vypíše sa, že chyba bola vo fragmente 0.

```
Choose client or server
1. Client
2. Server
Your choice: 1
Input ip of the server: 172.20.10.3
Input port: 50000
Type 1 to exit, anything to continue:
trying to connect to server
Connected to server
1 - exit
2 - switch client and server
3 - send message to server
Your choice: 3
Write 1 if you want to send message to server
Write 2 if you want to send file to server
Your choice: 1
Input fragment size: 5
Input your message: hello wolrd
Mistake in packet: 0
Fragment bol chybny, posielam znova
sended fragment number:  1  of  3
sended fragment number:  2  of  3
sended fragment number:  3  of  3
```

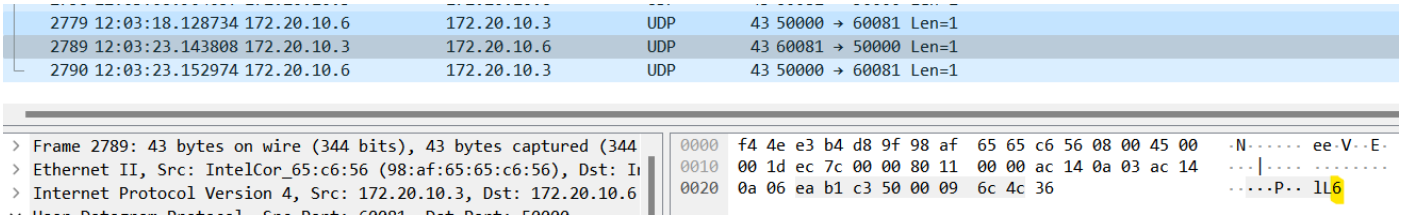
```
Choose client or server
1. Client
2. Server
Your choice: 2
Input port: 50000
Connected to client with address:  ('172.20.10.6', 55921)
1 - exit
Anything to continue
Your choice:
server is waiting for message from client
keep alive message
keep alive message
keep alive message
keep alive message
received 0 fragment with wrong crc
Number of fragments:  3
Size of fragment:  5
Size of message:  16
Received message:  hello wolrd
```

Skrypova Karolina ID: 123423

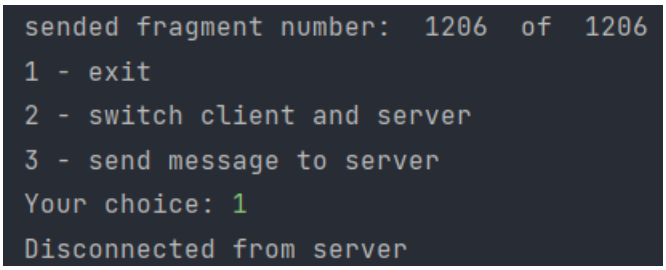
Po prijatí posledného fragmentu si server mohol vymeniť miesto s klientom, server si chcel vymeniť miesto, preto poslal klientovi správu typu 3 a po prijatí potvrdenia od klienta si vymenil miesto s klientom.

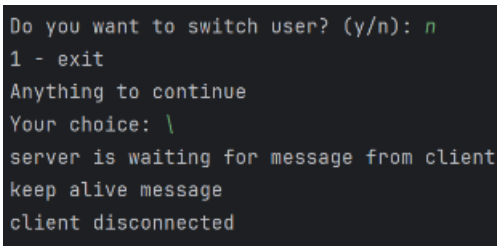


Potom inicializujú spojenie a server začne čakať na správu od klienta.



Na konci komunikácie klient odošle serveru správu o ukončení spojenia, poslednou správou bude odpoveď servera s potvrdením, po ktorej sa spojenie ukončí.





Dôležité časti kódu

Funkcia keep alive

Na spustenie funkcie keep alive pomocou funkcie spustenia vlákna vytvoríme nové vlákno, do ktorého sa bude posielať keep alive, na kontrolu keep alive máme aj globálny flag, ktorý ak je označený ako nepravdivý, vlákno na udržiavanie spojenia nebude fungovať, zvyčajne sa pri odosielaní súboru alebo textovej správy vlákno pre keep alive vypne.

Tiež v tejto funkcii sledujeme chyby, ak klient nedostane odpoveď do 30 sekúnd po odoslaní keep alive, vypíše sa chyba, potom sa komunikácia ukončí a používateľ bude požiadaný, aby si vybral novú IP alebo port, na ktorý sa môže pripojiť.

```python
# for keep alive
thread = None
thread_flag = False


def keep_alive(client_sock, server_address):
    global thread_flag
    global thread
    try:
        while True:
            if not thread_flag:
                return
            # sending keep alive message to server
            client_sock.sendto(str.encode("2"), server_address)
            # if no response from server for 30 sec, close connection
            client_sock.settimeout(30)
            # waiting for response from server
            data, address = client_sock.recvfrom(1472)
            if data.decode() == "2":
                time.sleep(5)
                continue
    except socket.timeout:
        print("no response from server")
        time.sleep(1)
        client_sock.close()
        menu_client()
        return
    except Exception as error:
        print(error)
        time.sleep(1)
        client_sock.close()
        menu_client()
        return


def start_thread(client_sock, server_address):
    global thread_flag
    global thread
    thread = threading.Thread(target=keep_alive, args=(client_sock, server_address))
    thread_flag = True
    thread.start()
```

Prepínanie medzi funkciou vysielača a prijímača

Prepínanie na strane servera:

Server má možnosť prepínať sa s klientom po každom prijatí textovej správy alebo súboru. Po tom, ako server prijme posledný fragment a výsledok sa zapíše do terminálu alebo uloží do súboru, server dostane možnosť rozhodnúť sa, či si s klientom vymení, klient počká na správu od servera so svojím rozhodnutím a pozastaví sa.

```python
    switch_user = input("Do you want to switch user? (y/n): ")
    if switch_user == "y":
        server_sock.sendto(str.encode("3"), client_address)
        data, client_address = server_sock.recvfrom(1472)
        if data.decode() == "3":
            print("Disconnectting from client")
            server_ip = socket.gethostbyname(socket.gethostname())
            server_sock.close()
            print("Closed server")
            time.sleep(6)
            print("Switching to client")
            create_client(client_address[0], port)
```

Na tento účel server najprv uzavrie socket a počká určitý čas, kým klient uzavrie svoj socket a vytvorí server, aby sa k nemu úspešne pripojil ako klient.

Prepínanie na strane klienta:

Keď si chce klient vymeniť miesto so serverom, pošle mu správu typu 3, po ktorej bude očakávať odpoveď rovnakého typu, čo bude znamenať, že server dal potvrdenie.

```
if choice == "3":
    send_to_server(client_sock, server_address)
    data, address = client_sock.recvfrom(1472)
    data = data.decode()
    if data == "3":
        client_sock.sendto(str.encode("3"), server_address)
        time.sleep(1)
        print("Disconnected from server")
        client_sock.close()
        time.sleep(3)
        print("Switching to server")
        create_server(port)
    start_thread(client_sock, server_address)
    continue
```

(Klient)

```
elif str(typ.decode()) == "3":
    server_sock.sendto(str.encode("3"), client_address)
    server_ip = socket.gethostbyname(socket.gethostname())
    #print("Server ip: ", server_ip)
    server_sock.close()
    time.sleep(2)
    print("Switching to client")
    time.sleep(3)
    create_client(client_address[0], port)
    return
```

(Server)

Potom sa klientský socket uzavrie, počká, kým server uzavrie svoj socket a vytvorí server, a spustí keep alive.

Odosielanie súborov a textových správ

Ak používateľ odošle textovú správu, najprv ju prevedie na bajty a potom nájde počet fragmentov pomocou funkcie math.ceil() Metóda math.ceil() v prípade potreby zaokrúhli číslo nahor na najbližšie celé číslo a vráti výsledok.

```
# message to bytes
message = str.encode(message)
# count number of fragments to send
number_of_fragments = math.ceil(len(message) / fragment_size)
```

Ak chce používateľ odoslať súbor, otvorí ho vo formáte bajtov a zapíše ho do premennej message

```
file_name = input("Input file name: ")
file_path = os.path.abspath(file_name)
try:
    file = open(file_path, "rb")
```

```
    file_size = os.path.getsize(file_name)
    number_of_fragments = math.ceil(file_size / fragment_size)
    message = file.read()
```

Pokiaľ neodošleme celú správu, vyberieme z nej prvých n bajtov a odošleme ich.

Predtým si ešte vytvoríme list s indexmi chybných fragmentov, aby sme vedeli, kam máme chyby doplniť.

```
    thread_flag = False
    if thread is not None:
        thread.join()
        thread = None
    packet_order = 0
    # sending number of fragments to server
    mistake_packets = make_mistake_packets(number_of_fragments)
    try:
        while message:
            client_sock.settimeout(30)
            if len(message) == 0:
                break
            part_message = message[:fragment_size]

            if type_file == 1:
                header = struct.pack("cHHHH", str.encode("4"), fragment_size, len(part_message), number_of_fragments,
                                     packet_order)
            else:
                header = struct.pack("cHHHH", str.encode("5"), fragment_size, len(part_message), number_of_fragments,
                                     packet_order)
            crc = binascii.crc_hqx(header + part_message, 0)
            if packet_order in mistake_packets:
                print("Mistake in packet: " + str(packet_order))
                crc += 1
            header = header + struct.pack("H", crc)
            client_sock.sendto(header + part_message, server_address)
            data, address = client_sock.recvfrom(1472)
            if data.decode() != "5":
                print("Fragment bol chybny, posielam znova")
                mistake_packets.remove(packet_order)
            else:
                packet_order += 1
                print("sended fragment number: ", packet_order, " of ", number_of_fragments)
                message = message[fragment_size:]
    except socket.timeout:
        print("no response from server")
        time.sleep(1)
        client_sock.close()
        menu_client()
```

Kód využíva vlastný protokol, v ktorom sú správy rozdelené na fragmenty, pričom ku každému je pripojená hlavička obsahujúca informácie o danom fragmente a hodnotu CRC (Cyclic Redundancy Check) na kontrolu chýb. Okrem toho kód spracováva potenciálne chyby v konkrétnych paketoch a pokúša sa ich opätovne odoslať.

Na odoslanie dát spolu s hlavičkou použijeme funkciu z knižnice struct:

struct.pack je funkcia v module struct jazyka Python, ktorá sa používa na zabalenie hodnôt do binárneho dátového reťazca podľa zadaného formátu.

Prijímanie súborov a textových správ

Vo funkcii načítania fragmentov server najprv rozbalí hlavičku, ktorú prijal počas prenosu, konkrétne prvých 12 bajtov, a potom definuje premenné ako typ, veľkosť fragmentu, veľkosť správy, počet fragmentov, slušné číslo a crc.

```
unpacked_data = struct.unpack("cHHHHH", data[:12])
type_header, fragment_size, part_message_size, number_of_fragments, packet_order, crc = unpacked_data
```

Potom definujeme údaje, ktorých správnosť musíme skontrolovať, a to konkrétne hlavičku bez crc spolu s datami, potom porovnáme nové crc s tým, ktoré bolo odoslané v hlavičke, a ak je rovnaké, potom je fragment správny, v opačnom prípade fragment obsahoval chybu a server pošle klientovi žiadosť o opätovné odoslanie fragmentu.

```
data_for_check = data[:10]+data[12:]
check_crc = binascii.crc_hqx(data_for_check, 0)
if check_crc == crc:
    server_sock.sendto(str.encode("5"), client_address)
    result_list.append([packet_order, data[12:].decode()])
    if len(result_list) == number_of_fragments:
        break
else:
    server_sock.sendto(str.encode("7"), client_address)  #stop and wait
    print("received " + str(packet_order) + " fragment with wrong crc")
    continue
```

Tiež keď dostaneme posledný fragment, ukončíme cyklus a prejdeme k vypísaniu výsledku. ak sme predtým poslali text, zapíšeme ho do terminálu, ak sme poslali súbor, napíšeme do terminálu, kam chceme tento súbor uložiť, a potom sa údaje zapíšu do súboru.

```
print("Number of fragments: ", number_of_fragments)
print("Size of fragment: ", fragment_size)
print("Size of message: ", size_sum)
size_sum = 0
if message_type == 4:
    result_string = "".join(result_list[i][1] for i in range(len(result_list)))
    print("Received message: ", result_string)
else:
    while True:
        file_path = input("Write path to save file: ")
        if not os.path.exists(file_path):
            print("Path does not exist")
            continue
        break
    file_name = file_path + "\\" + file_name
    with open(file_name, "wb") as file:
        for fragment in result_list:
            file.write(fragment[1])
        file.close()
    print("Received file: ", file_name)
    print("path: ", file_name)
```

Použité knižnice

binascii - pre funkciu cps, pomocou ktorej kontrolujeme správnosť činnosti funkcie

math - pre funkciu delenia so zaokrúhľovaním na väčšie číslo

os.path - na vypísanie absolútnej cesty k súboru

socket - na vytváranie soketov a komunikáciu medzi 2 používateľmi

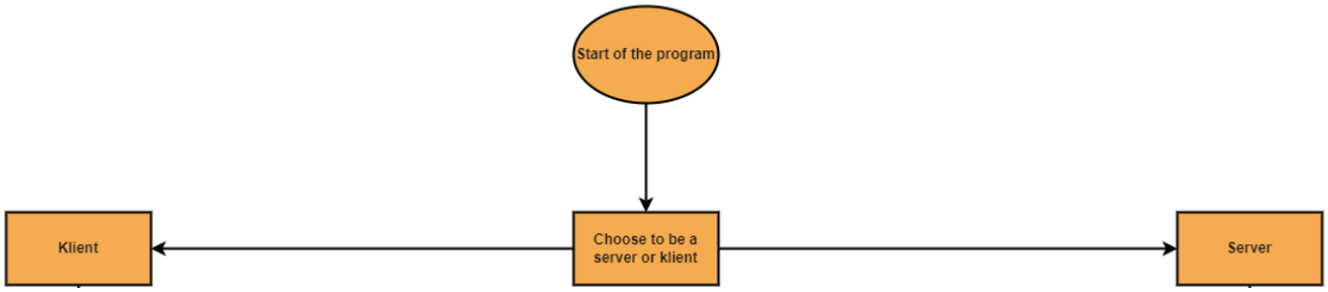
struct - na spájanie údajov a prevod do binárneho tvaru

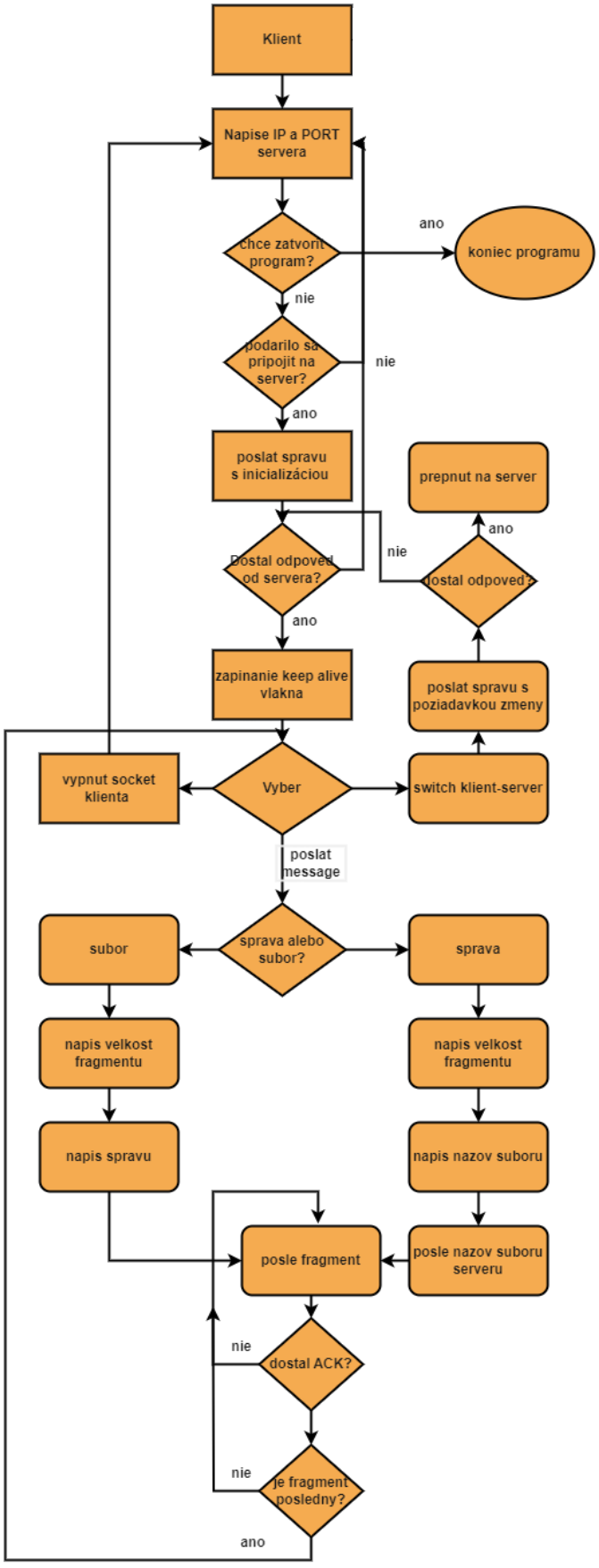
threading - na vytváranie a kontrolu vlákien

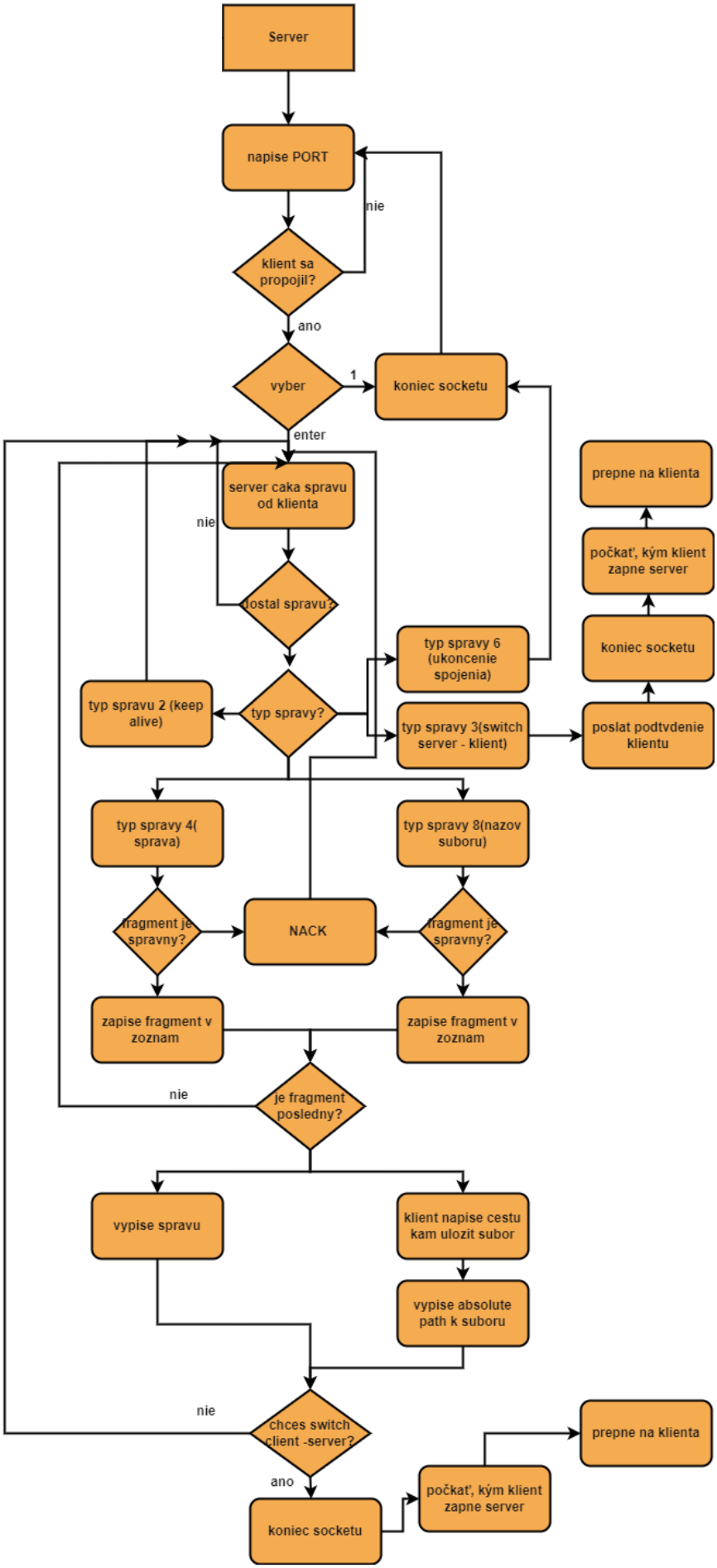
time - pre funkciu sleep a možnosť pozastaviť vlákna na určitý čas.

Diagramy

Diagramy sa oproti návrhu riešenia veľmi nezmenili, ale doplnil som do nich niektoré špecifiká a tiež som sa rozhodol rozdeliť diagram na 2 časti, časť pre klienta a časť pre server.




Klient
Napise IP a PORT servera
chce zatvorit program?
ano
koniec programu
nie
podarilo sa pripojit na server?
nie
ano
poslat spravu s inicializáciou
prepnut na server
Dostal odpoved od servera?
nie
ano
dostal odpoved?
ano
zapinanie keep alive vlakna
poslat spravu s poziadavkou zmeny
vypnut socket klienta
Vyber
switch klient-server
poslat message
subor
sprava alebo subor?
sprava
napis velkost fragmentu
napis velkost fragmentu
napis spravu
napis nazov suboru
posle fragment
posle nazov suboru serveru
nie
dostal ACK?
nie
je fragment posledny?
ano



Skrypova Karolina ID: 123423