

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Zsolt Kiss

Počítačové a komunikačné siete

UDP Komunikátor

Zadanie 2

Piatok 8:00-9:50

Ing. Matej Janeba

Obsah

Analýza problematiky	3
Návrh riešenia	3
Vlastná hlavička.....	5
ARQ metóda – Stop and Wait	6
Výpočet kontrolnej sumy CRC	6
Udržiavanie spojenia.....	6
Typy možných rámcov	7
Vývojový diagram	9
Opis používateľského rozhrania	10
Režim servera.....	10
Režim klienta	11
Zmeny oproti návrhu.....	14
O zdrojovom kóde.....	14
Globálne premenné	14
Dôležité funkcie	15
Pomocné funkcie.....	16
Záver	17

Analýza problematiky

Zadaním úlohy je implementácia komunikátora s použitím vlastného protokolu nad protokolom UDP transportnej vrstvy sieťového modelu TCP/IP. Komunikátor bude umožniť komunikáciu dvoch účastníkov v lokálnej sieti Ethernet, teda prenos správ medzi počítačmi (uzlami).

O protokole UDP vieme, že je:

Nespol'ahlivý – nezaručí, že fragmenty dorazia bez poškodenia a iba raz.

Neusporiadaný – poradie doručených fragmentov nemusí byť rovnaké ako poradie, v ktorom boli odoslané.

Nespojový – nevieme, či druhá strana je stále prítomná.

Náš protokol musí byť:

Spol'ahlivý – po skončení prenosu, musí byť celá správa kompletne preposlaná, všetky fragmenty musia byť nepoškodené.

Spojový – chceme vedieť, či je druhá strana stále prítomná, a ak dôjde k odpojeniu, chceme byť informovaný o prerušení spojenia.

Nefragmentovaný (na linkovej vrstve) – musíme zaručiť, aby rámec nebol ďalej fragmentovaný v rámci Ethernetu.

Návrh riešenia

Ako riešime problémy s UDP protokolom vo vlastnej implementácii?

Spol'ahlivosť

Na overenie doručenia a integrity fragmentov použijeme pozitívne (ACK) a negatívne potvrdenia (NACK). Po odoslaní fragmentu bude odosielajúca strana čakať na potvrdenie zo servera. Ak príde kladné potvrdenie, odošle ďalší fragment, v prípade negatívneho potvrdenia opäť požiada o preposlanie chybného fragmentu.

Každý rámec bude obsahovať 32 bitový CRC checksum, ktorý bude slúžiť na to, aby sme vedeli overiť správnosť obsahu daného rámca.

Spojovosť

Po úspešnom pripojení medzi klientom a serverom klient po prvej odoslanej správe odošle správu Keep Alive, aby udržal spojenie so serverom.

Server pošle späť pozitívne potvrdenie, čo znamená „som tu, stále počúvam“. Ak server v určitom čase nedostane od klienta žiadnu správu, považuje sa spojenie za uzavreté.

Tak isto, ak klient nedostane od servera kladné potvrdenie za určitý čas, považuje sa spojenie za uzavreté.

Nefragmentovanosť

Maximálna veľkosť Ethernetovej hlavičky je 1500 bajtov. IP hlavička má maximálnu veľkosť 60 bajtov a UDP hlavička je 8 bajtová. Môj program som vyskúšal spustiť medzi dvomi zariadeniami v domácej sieti, kde som overil, že Zostáva nám maximálne 1458 bajtov, ktorých môžeme využiť na rámce nášho protokolu. Táto veľkosť predstavuje maximálnu veľkosť celého rámca, ktorý môžeme odoslať/priať bez toho, aby bol rámec ďalej fragmentovaný na linkovej vrstve. Náš protokol má vlastnú 13 bajtovú hlavičku, ktorú ešte musíme odpočítať od celkovej veľkosti, takže dostaneme výsledok, ktorým je maximálna veľkosť nespracovaných dát na fragment. Z toho vyplýva, že odosielaťca strana nemôže nastaviť veľkosť fragmentu na viac ako 1445 bajtov.

Vlastná hlavička

Náš protokol bude mať vlastnú hlavičku, ktorá má 5 atribútov:

sequence_number	fragment_count	fragment_size	packet_type	CRC
3 bajty	3 bajty	2 bajty	1 bajt	4 bajty

sequence_number je poradové číslo rámca, slúži na znovuzoskupenie správy.

fragment_count je počet fragmentov, na ktoré je rozdelená celá správa. Toto je dôležitá informácia, aby server vedel, koľko rámcov má prijať.

fragment_size nám hovorí, koľko bajtov dát obsahuje aktuálny rámec. Táto veľkosť je definovaná klientom a je to veľkosť **data** bez hlavičky. V prípade informačnej správy táto hodnota nie je reálna veľkosť rámca, je iba klientom zadefinovaná veľkosť dátových fragmentov (bez hlavičky). Príklad.: Odosielajúca strana nastaví veľkosť fragmentov na 1419 bajtov, ale správa “Hello, world”, ktorá nasleduje po informačnej správe sa zmestí do 12 bajtov.

packet_type určuje typ rámca:

- 0 - INF (informačná správa)
- 1 - ACK (pozitívne potvrdenie)
- 2 - NACK (negatívne potvrdenie)
- 3 - DAT (dátový prenos)
- 4 - KPA (Keep Alive)
- 5 – FIN (finančná správa)

CRC je vypočítaný z celého rámca, bez atribútu CRC. (bereme do úvahy bajty od 0 až 9 a bajty od 13 až do konca rámca.

ARQ metóda – Stop and Wait

Pre môj návrh som zvolil metódu ARQ Stop and wait. Táto metóda spočíva v tom, že odosielateľ odošle jeden rámec, zastaví sa a čaká na potvrdenie od prijímajúcej strany. Prijímajúca strana príjme rámec a ak je rámec v poriadku, pošle pozitívne potvrdenie. Ak je rámec prijatý, poškodený požiada odosielateľa o opätovné odoslanie rámca. Keď odosielateľ dostane potvrdenie o prijatí rámca, rozhodne sa, či pošle ďalší rámec v poradí, alebo pošle rámec, ktorý bol poškodený ešte raz.

Výpočet kontrolnej sumy CRC

Na výpočet CRC budem používať knižnicu zlib, ktorá poskytuje funkciu crc32 na výpočet 32 bitového kontrolného súčtu. Táto metóda používa generujúci polynóm $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$.

Udržiavanie spojenia

Po prenose prvej správy, klient posiela Keep Alive správy serverovi. Server na tie správy odpovedá potvrdením alebo finalizačnou správou. Keďže v mojom programe server nemôže priamo poslať správy, preto musí čakať na ďalšiu Keep Alive správu, na čo už môže poslať finalizačnú správu, ak chce. Klient tiež môže poslať finalizačnú správu namiesto Keep Alive správy. V oboch prípadoch, nie je nutné potvrdenie doručenie finalizačnej správy.

Typy možných rámcov

INF rámec - posiela klient serverovi

sequence_number	fragment_count	fragment_size	packet_type	CRC	data
5	12	3	0	4083405295	b'<F>'

Pri informačnej správe, atribút **sequence_number** hovorí o tom, koľko rámcov bude obsahovať meno súboru v dátovej časti. Atribút **fragment_count** je celkový počet rámcov, ktoré budú prenášané po informačnej správe. Klient rozdelil správu po 3 bajtoch. **packet_type** 0 označuje, že ide o informačnú správu. **data** v informačnej správe nám hovorí o tom, či nasleduje prenos textovej správy alebo prenos súboru. Ak ide o textovú správu, obsah hárku **data** v informačnej správe vždy bude b'<T>' a sequence_number bude 0, keďže tam nemáme názov súboru. Ak ide o prenos súboru, tak hárak **data** bude obsahovať b'<F>'.

ACK na INF rámec - posiela server klientovi

sequence_number	fragment_count	fragment_size	packet_type	CRC	data
5	12	3	1	0	b'<F>'

packet_type 1 označuje, že ide o pozitívne potvrdenie (ACK). **CRC** je nastavený na 0, lebo klientská strana nebude kontrolovať CRC, takže nie je potrebné ho vypočítať. **sequence_number**, **fragment_count**, **fragment_size** a **data** sú to isté, aké poslal klient.

DAT rámec - posiela klient serverovi

sequence_number	fragment_count	fragment_size	packet_type	CRC	data
1	12	3	3	3136520246	b'sma'

sequence_number už obsahuje reálne poradové číslo rámcu. **packet_type** 3 označuje, že ide o dátový prenos. **data** už obsahuje časť správy.

ACK na DAT rámeč - posiela server klientovi

sequence_number	fragment_count	fragment_size	packet_type	CRC	data
1	12	3	1	0	b"

NACK na DAT rámeč - posiela server klientovi

sequence_number	fragment_count	fragment_size	packet_type	CRC	data
1	12	3	2	0	b"

packet_type určuje, či ide o pozitívne alebo negatívne potvrdenie. CRC je nastavený na 0, lebo klientská strana nebude kontrolovať CRC, takže nie je potrebné vypočítať. **sequence_number**, **fragment_count** a **fragment_size** sú tie isté, aké poslal klient v dátovom rámci. **data** neobsahuje žiadne údaje.

KPA rámeč - posiela klient serverovi

sequence_number	fragment_count	fragment_size	packet_type	CRC	data
0	0	0	4	0	b"

ACK na KPA rámeč - posiela server klientovi

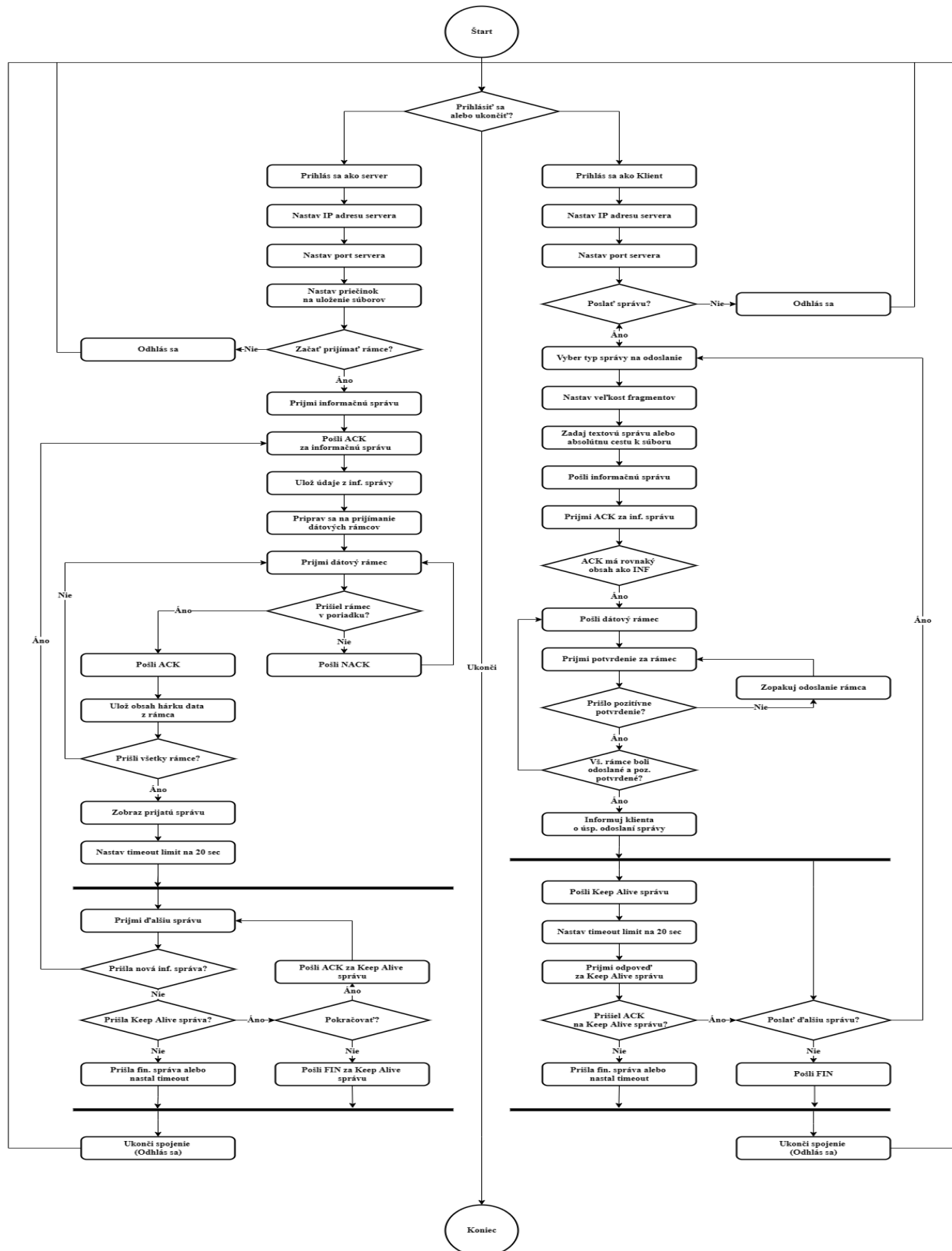
sequence_number	fragment_count	fragment_size	packet_type	CRC	data
0	0	0	1	0	b"

FIN rámeč – posiela klient alebo server

sequence_number	fragment_count	fragment_size	packet_type	CRC	data
0	0	0	5	0	b"

Ak niektorá strana dostane FIN správu, znamená že druhá strana chce ukončiť komunikáciu a už zatvoril socket na svojej strane.

Vývojový diagram



Opis používateľského rozhrania

Na začiatku používateľ má 3 možnosti.

- (0) – Quit the application = Ukončiť aplikáciu
- (1) – Log in as server = Prihlásiť sa ako server
- (2) – Log in as client = Prihlásiť sa ako klient

Používateľ zadá hodnotu podľa svojej potreby a potvrdí svoju voľbu stlačením tlačidla ENTER.

Režim servera

Ak používateľ vyberá rolu servera, musí si najprv nastaviť IP adresu a port, na ktorom bude počúvať a absolútnu cestu k adresáru, kam si bude ukladať súbory prijaté od klienta. Po nastavení týchto údajov sa ho program opýta, či naozaj chce začať prijímať rámce. V tomto momente sa používateľ môže stále odhlásiť, ale ak sa rozhodne spustiť server, musí počkať na prvú správu, dovtedy nebude možné sa odhlásiť.

```
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.9_3.9.2544.0_x64_qbz5n2kfra8p0\python3.9.exe
Choose from the options:
(0) - Quit application
(1) - Log in as server
(2) - Log in as client
1
>>> SERVER <<<

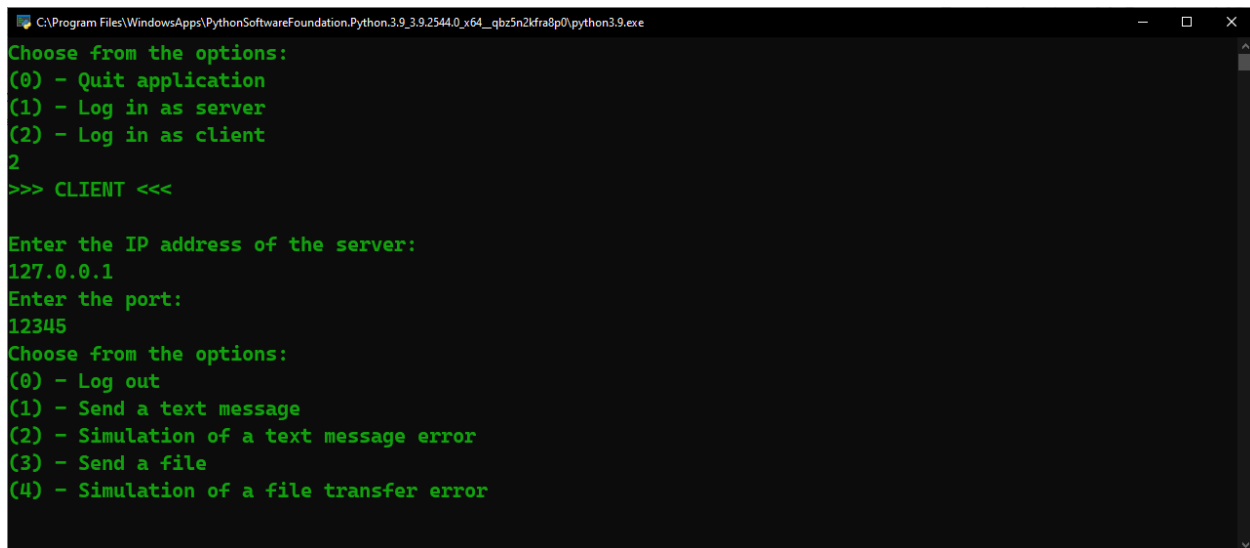
Enter the IP address of the server:
127.0.0.1
Enter the port:
12345
Enter the name of the folder where you want to save the files:
C:\Users\destr\PycharmProjects\PKS_Zadanie2\server
The files will be saved in the directory C:\Users\destr\PycharmProjects\PKS_Zadanie2\server
Choose from the options:
(0) - Log out
(1) - Start the server (Wait for the first message)
1
>>> Server is running <<<
```

Režim klienta

Ak si používateľ vyberie rolu klienta, musí nastaviť IP adresu a port servera, s ktorým bude počas relácie komunikovať. Následne môže vybrať typ správy, akú chce poslať alebo sa môže odhlásiť.

Možnosti sú:

- (0) – Log out = Odhlásiť sa
- (1) – Send a text message = Poslať textovú správu
- (2) – Simulation of a text message error = Simulácia chyby textovej správy
- (3) – Send a file = Poslať súbor
- (4) – Simulation of a file transfer error = Simulácia chyby prenosu súboru



```
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.9.3.2544.0_x64__qbz5n2kfra8p0\python3.9.exe
Choose from the options:
(0) - Quit application
(1) - Log in as server
(2) - Log in as client
2
>>> CLIENT <<<

Enter the IP address of the server:
127.0.0.1
Enter the port:
12345
Choose from the options:
(0) - Log out
(1) - Send a text message
(2) - Simulation of a text message error
(3) - Send a file
(4) - Simulation of a file transfer error
```

Ak používateľ zvolí možnosť poslať textovú správu alebo simulovať chybu pri prenose súboru, musí zadať veľkosť fragmentov a následne absolútnu cestu k súboru.

```
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.9_3.9.2544.0_x64_qbz5n2kfra8p0\python3.9.exe
Choose from the options:
(0) - Quit application
(1) - Log in as server
(2) - Log in as client
2
>>> CLIENT <<<

Enter the IP address of the server:
127.0.0.1
Enter the port:
12345
Choose from the options:
(0) - Log out
(1) - Send a text message
(2) - Simulation of a text message error
(3) - Send a file
(4) - Simulation of a file transfer error
1
Enter the size of the fragments. 13 bytes will be automatically added to the value.
4
Enter a text message
Toto je textova sprava.
SENT : [>] sequence_number 0 | fragment_count 6 | fragment_size 17 bytes | INF | CRC 489210236 | data b'T'
RECEIVED: [✓] sequence_number 0 | fragment_count 6 | fragment_size 17 bytes | ACK | CRC 0 | data b'T'
SENT : [>] sequence_number 1 | fragment_count 6 | fragment_size 17 bytes | DAT | CRC 369487864 | data b'Toto'
RECEIVED: [✓] sequence_number 1 | fragment_count 6 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 2 | fragment_count 6 | fragment_size 17 bytes | DAT | CRC 3637183432 | data b' je '
RECEIVED: [✓] sequence_number 2 | fragment_count 6 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 3 | fragment_count 6 | fragment_size 17 bytes | DAT | CRC 3307010609 | data b'text'
RECEIVED: [✓] sequence_number 3 | fragment_count 6 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 4 | fragment_count 6 | fragment_size 17 bytes | DAT | CRC 2186878704 | data b'ova '
RECEIVED: [✓] sequence_number 4 | fragment_count 6 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 5 | fragment_count 6 | fragment_size 17 bytes | DAT | CRC 1033987377 | data b'spra'
RECEIVED: [✓] sequence_number 5 | fragment_count 6 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 6 | fragment_count 6 | fragment_size 16 bytes | DAT | CRC 2533554357 | data b'va.'
RECEIVED: [✓] sequence_number 6 | fragment_count 6 | fragment_size 16 bytes | ACK | CRC 0 | data b''
>>> The message 'Toto je textova sprava.' has been sent successfully <<<
```

Ak používateľ zvolí možnosť poslať súbor alebo simulovať chybu pri prenose súboru, musí zadať veľkosť fragmentov a absolútnu cestu k súboru.

```
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.9_3.9.2544.0_x64__qbz5n2kfra8p0\python3.9.exe
Choose from the options:
(0) - Quit application
(1) - Log in as server
(2) - Log in as client
2
>>> CLIENT <<<

Enter the IP address of the server:
127.0.0.1
Enter the port:
12345
Choose from the options:
(0) - Log out
(1) - Send a text message
(2) - Simulation of a text message error
(3) - Send a file
(4) - Simulation of a file transfer error
3
Enter the size of the fragments. 13 bytes will be automatically added to the value.
4
Enter the file name (enter the full path):
C:\Users\destr\PycharmProjects\PKS_Zadanie2\client\smallFile.bin
>>> File C:\Users\destr\PycharmProjects\PKS_Zadanie2\client\smallFile.bin will be sent to the client. <<<
SENT : [>] sequence_number 4 | fragment_count 9 | fragment_size 17 bytes | INF | CRC 3901434143 | data b'F'
RECEIVED: [✓] sequence_number 4 | fragment_count 9 | fragment_size 17 bytes | ACK | CRC 0 | data b'F'
SENT : [>] sequence_number 1 | fragment_count 9 | fragment_size 17 bytes | DAT | CRC 1610090771 | data b'smal'
RECEIVED: [✓] sequence_number 1 | fragment_count 9 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 2 | fragment_count 9 | fragment_size 17 bytes | DAT | CRC 849415122 | data b'lFil'
RECEIVED: [✓] sequence_number 2 | fragment_count 9 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 3 | fragment_count 9 | fragment_size 17 bytes | DAT | CRC 1534929988 | data b'e.bi'
RECEIVED: [✓] sequence_number 3 | fragment_count 9 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 4 | fragment_count 9 | fragment_size 14 bytes | DAT | CRC 3781768811 | data b'n'
RECEIVED: [✓] sequence_number 4 | fragment_count 9 | fragment_size 14 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 5 | fragment_count 9 | fragment_size 17 bytes | DAT | CRC 4199051360 | data b'It i'
RECEIVED: [✓] sequence_number 5 | fragment_count 9 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 6 | fragment_count 9 | fragment_size 17 bytes | DAT | CRC 1911230392 | data b's a '
RECEIVED: [✓] sequence_number 6 | fragment_count 9 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 7 | fragment_count 9 | fragment_size 17 bytes | DAT | CRC 3075326800 | data b'smal'
RECEIVED: [✓] sequence_number 7 | fragment_count 9 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 8 | fragment_count 9 | fragment_size 17 bytes | DAT | CRC 1806044292 | data b'l fi'
RECEIVED: [✓] sequence_number 8 | fragment_count 9 | fragment_size 17 bytes | ACK | CRC 0 | data b''
SENT : [>] sequence_number 9 | fragment_count 9 | fragment_size 16 bytes | DAT | CRC 3852218483 | data b'le.'
RECEIVED: [✓] sequence_number 9 | fragment_count 9 | fragment_size 16 bytes | ACK | CRC 0 | data b''
>>> The file 'smallFile.bin' has been sent successfully <<<
```

Zmeny oproti návrhu

Oproti návrhu som zmenil ako funguje informačná správa. V návrhu som to mal tak, že `sequence_number` bude slúžiť na to, aby server vedel, koľko bajtov dát má očakávať spolu. Vtedy som ešte nerátal s tým, že keď posielame súbor, tak treba fragmentovať aj názov súbora. Vtedy informačná správa obsahoval aj názov súbora v hárku data, bez ohľadu na jej veľkosť. V implementácii pri informačnej správe `sequence_number` obsahuje počet dodatočných rámcov, v ktorých je fragmentovaný názov súbora. Taktiež som zmenil ako funguje Keep Alive, teraz ak bola poslaná aspoň jedna správa, obe strany posielajú finančnú správu druhej strane, ak chcú končiť komunikáciu. Dokumentácia je aktualizovaná, a všetko je napísané tak, ako to v programe funguje. Tak isto som zmenil aj vývojový diagram.

O zdrojovom kóde

Globálne premenné

Sekcia SWITCHES

Tieto premenné sú editovateľné, ich zmena má vplyv ako sa zobrazujú prijaté a odoslané rámce.

`SHOW_ATTRIBUTES` – ak je nastavený na True, zobrazí atribúty rámcov

`SHOW_RAW_DATA` – ak je nastavený na True, zobrazí dátovú časť rámca.

`SHOW_ATTRIBUTES` musí byť True, aby tento switch mal efekt.

`CLIENT_SHOW_KPAs_AND_FINs` – ak je nastavený na True, klientská strana zobrazí odoslané a prijaté Keep Alive správy.

Sekcia EVENTS

Tu sú deklarované udalosti, ktoré sú potrebné na správne fungovanie Keep Alive-u.

Sekcia SIZES

`MAX_DATA_SIZE_IN_BYTES` – maximálna veľkosť rámca s hlavičkou v bajtoch

CUSTOM_HEADER_SIZE_IN_BYTES – veľkosť vlastnej hlavičky

TIMEOUT_IN_SECONDS – čas timeoutu v sekundách

KPA_SENDING_FREQUENCY_IN_SECONDS – pravidelnosť posielania Keep Alive správy v sekundách

DAMAGE_EVERY_NTH_PACKET – pri simulácii chyby, každý n-tý rámec bude poškodený

Sekcia PACKET_TYPES

Tu sú deklarované jednotlivé typy rámcov ako čísla.

Sekcia BUFFERS

Tu sú deklarované pomocné premenné, používané pri vláknach. Needitovať.

Dôležité funkcie

def calculate_fragment_count(file_size, fragment_size)

Táto funkcia vypočíta na koľko fragmentov rozdelíme správu. Parameter file_size je veľkosť celého súboru v bajtoch, fragment_size je klientom zadaná veľkosť fragmentov.

def calculate_data_length(buffer, file_size, fragment_size)

Táto funkcia vypočíta aktuálnu veľkosť dátovej časti fragmentu, ku ktorým bude vždy pripočítaný veľkosť vlastnej hlavičky, čiže 13 bajtov, a to bude reálna veľkosť fragmentu.

def create_custom_header(sequence_number, fragment_count, fragment_size, packet_type)

Táto funkcia vytvorí časť hlavičky, bez atribútov CRC a data, ktoré budú neskôr pred odoslaním priložené k hlavičke.

def decode_data(data):

Táto funkcia slúži na dekodovanie hlavičky z bajtov.

def server_logout()

Táto funkcia sa spúšťa na vlastnom vlákne, monitoruje vstup používateľa počas komunikácie a odhlási sa zo servera, ak používateľ zadá 0.

def configure_server()

Táto funkcia slúži na nastavenie serverskeho režimu pred jeho spustením. Nastaví sa IP adresa, port a adresár na uloženie súborov.

def server(sock, path, server_input_thread)

Hlavná funkcia, kde server vykoná všetky svoje úlohy, prijíma rámce, posiela potvrdenia.

def client_keep_alive(server_ip, server_port, sock)

Táto funkcia sa spúšťa na vlastnom vlákne, má za úlohu posielat' Keep Alive správy prijímať potvrdenia za nich, tak tiež aj posielat' alebo prijať FIN správu.

def configure_client()

Táto funkcia slúži na nastavenie klientskeho režimu pred jeho spustením. Nastaví sa IP adresa servera a port.

def client(server_ip, server_port, sock)

Hlavná funkcia, kde klient vykoná všetky svoje úlohy okrem udržiavanie spojenia. Tu v tejto funkcii klient posiela rámce a prijíma potvrdenia za nich.

Pomocné funkcie

def validate_ip_address(address)

Táto funkcia skontroluje formát zadanej IP adresy.

def input_IP()

Táto funkcia slúži na načítanie IP adresy cez príkazový riadok.

def input_port()

Táto funkcia slúži na načítanie portu cez príkazový riadok.

def create_directory()

Táto funkcia vytvorí adresár pre server kam budú uložené súbory.

def set_fragment_size()

Táto funkcia slúži na načítanie veľkosti fragmentu cez príkazový riadok.

def has_the_same_header_except_flag(sent_packet, received_packet, flag)

Táto funkcia slúži na porovnávanie dvoch hlavičiek. Skontrolujeme atribúty okrem `packet_type` a `data`.

```
def packet_format(packet)
```

Táto funkcia slúži iba na výpis rámca vo formáte String.

Záver

Úlohou bolo navrhnutie komunikátora s využitím UDP datagramov. Program umožňuje jednosmernú komunikáciu z klienta na server. Pri odosielaní správ umožňuje zaslanie rámce s umelou chybou. Myslím si, že môj program splní všetky požiadavky. Toto zadanie ma veľa naučilo o komunikáciách medzi dvoma stranami, a lepšie pracovať so socketmi v jazyku Python.