Bitarovský UDP komunikátor

Obsah

Návrh riešenia

Zadanie

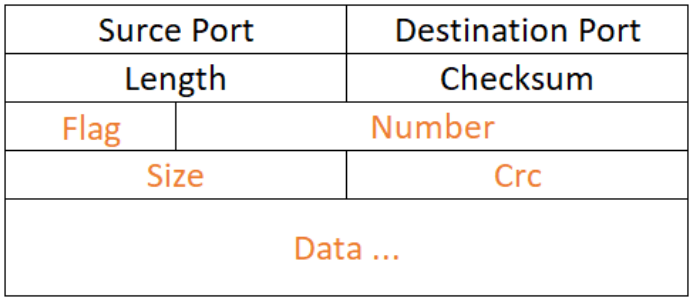
Navrhnite a implementujte program s použitím vlastného protokolu nad protokolom UDP (User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP. Program umožní komunikáciu dvoch účastníkov v lokálnej sieti Ethernet, teda prenos textových správ a ľubovoľného súboru medzi počítačmi (uzlami).

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 5-20s pokiaľ používateľ neukončí spojenie. Odporúčame riešiť cez vlastne definované signalizačné správy.

Vlastná hlavička



čierne sú atribúty UPD hlavičky a oranžové atribúty sú moja vlastná hlavička a dáta kt. budem "mnou vytvorený paket" obsahovať

UDP header

Source Port, Destination Port, Length a Checksum sú polia UDP hlavičky, ktoré hlavička obsahuje predvolene. Táto hlavička samo o sebe má 8 bajtov.

Vlastné Flags

> Plánovaná veľkosť 1B

Je premenná obsahujúca int v hodnote 0-255 aby bolo možné uložiť túto hodnotu do 1B. Flag je určený typ správy. Možné flags sú nasledovné:

- SYN → určuje inicializáciu spojenia

- ACK → určuje pozitívne potvrdenie
- NACK → určuje negatívne potvrdenie
- KA → určuje keep alive
- RST → určuje ukončenie spojenia
- START → určuje začiatok prenosu dát
- TEXT → určuje posielanie fragmentu textovej správy alebo fragmentov obsahu súboru
- FILE → určuje posielanie názvu súboru

Number

> Plánovaná veľkosť 3B

Poradové číslo predstavuje poradie fragmentu pre daný súbor. Je potrebné pre zostavenie správy v prípade ak fragmenty prídu v inom poradí v akom boli odoslané.

Size

> Plánovaná veľkosť 2B

Označuje veľkosť, akú má moja hlavička + data v bajtoch. Je možné, že tento atribút bude môcť byť z finálnej implementácie odstránený alebo nahradený iným atribútom.

CRC

> Plánovaná veľkosť 2B

Časť CRC (Cyclic redundancy check) je môj vlastný kontrolný súčet hlavičky a dát. Je vypočítaný klientom a následne zapísaný do hlavičky. Pri výpočte zohľadňujeme, že hodnota tejto premennej je nula.

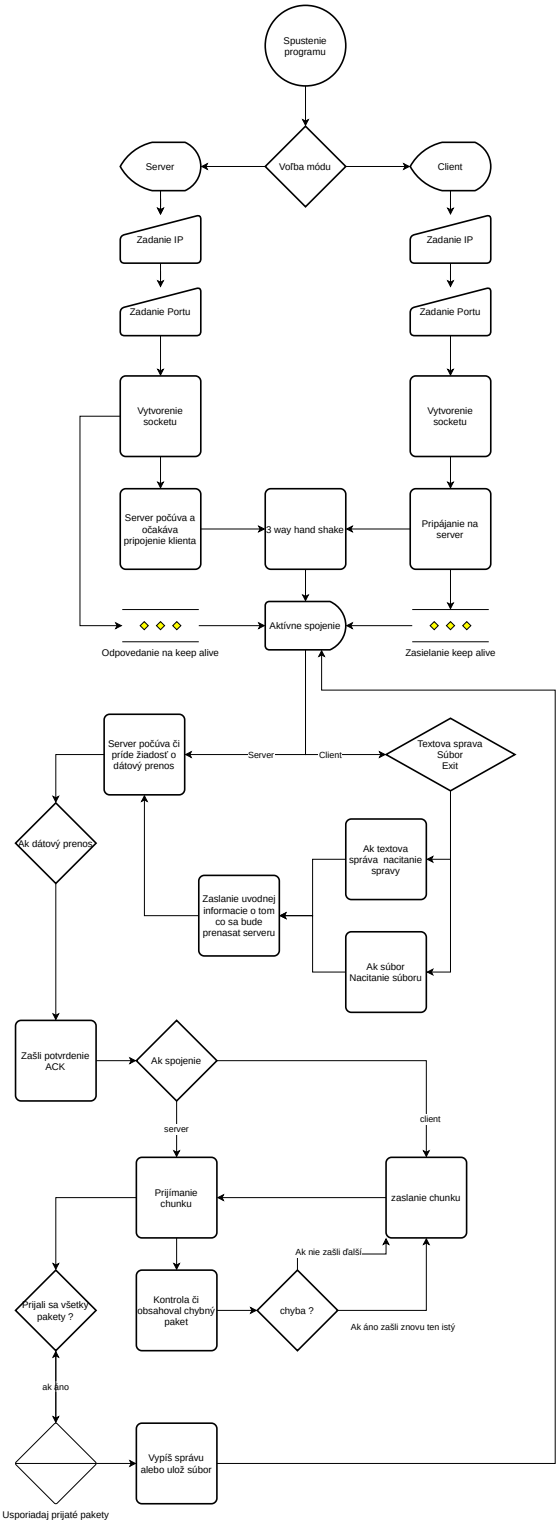
Server túto hodnotu vypočíta samostatne a následne overí či sedí s hodnotou, kt obsahuje doručený fragment. Ak sa hodnoty nezhodujú server označí fragment za chybný.

Veľkosť pre dáta

```
# MAX_DATA_SIZE = ETH_II_PAYLOAD - IP_HEADER_LEN - UDP_HEADER_LEN - MY_HEADER
MAX_DATA_SIZE = 1500 - 20 - 8 - 8
```

Teda výsledná veľkosť pre dáta, kt bude môcť niesť môj packet bez toho aby bol rozfragmentovaný na linkovej vrstve je 1464B

Diagram programu



Metóda vnesenia chyby do prenosu

Chyba sa do prenosu bude vkladať takým spôsobom, že sa ukladaná CRC hodnota na strane klienta zmenší o 1 a to bude znamenať, že check sum vypočítaný na strane servera sa nezhoduje s doručeným a server tento fragment vyhodnotí ako chybný.

Takto umelo sa budú (v prípade potreby) poškodzovať pakety nesúce informácie v dátovej časti.

Metóda kontrolného súčtu

Ako metódu kontroly som zvolil CRC (Cyclic Redundancy Check), teda kontrolu cyklickým kódom. Na tento výpočet sa použije prevzatá implementácia z nasledovného zdroja:

https://stackoverflow.com/questions/35205702/calculating-crc16-in-python

Ten atribút je v mojej hlavičke ako samostatná časť vid. tu.

Metóda ARQ

Selective Repeat ARQ je prístup, kt opätovne vyžiada iba pakety, ktoré boli označené ako chybné, alebo nedoručené.

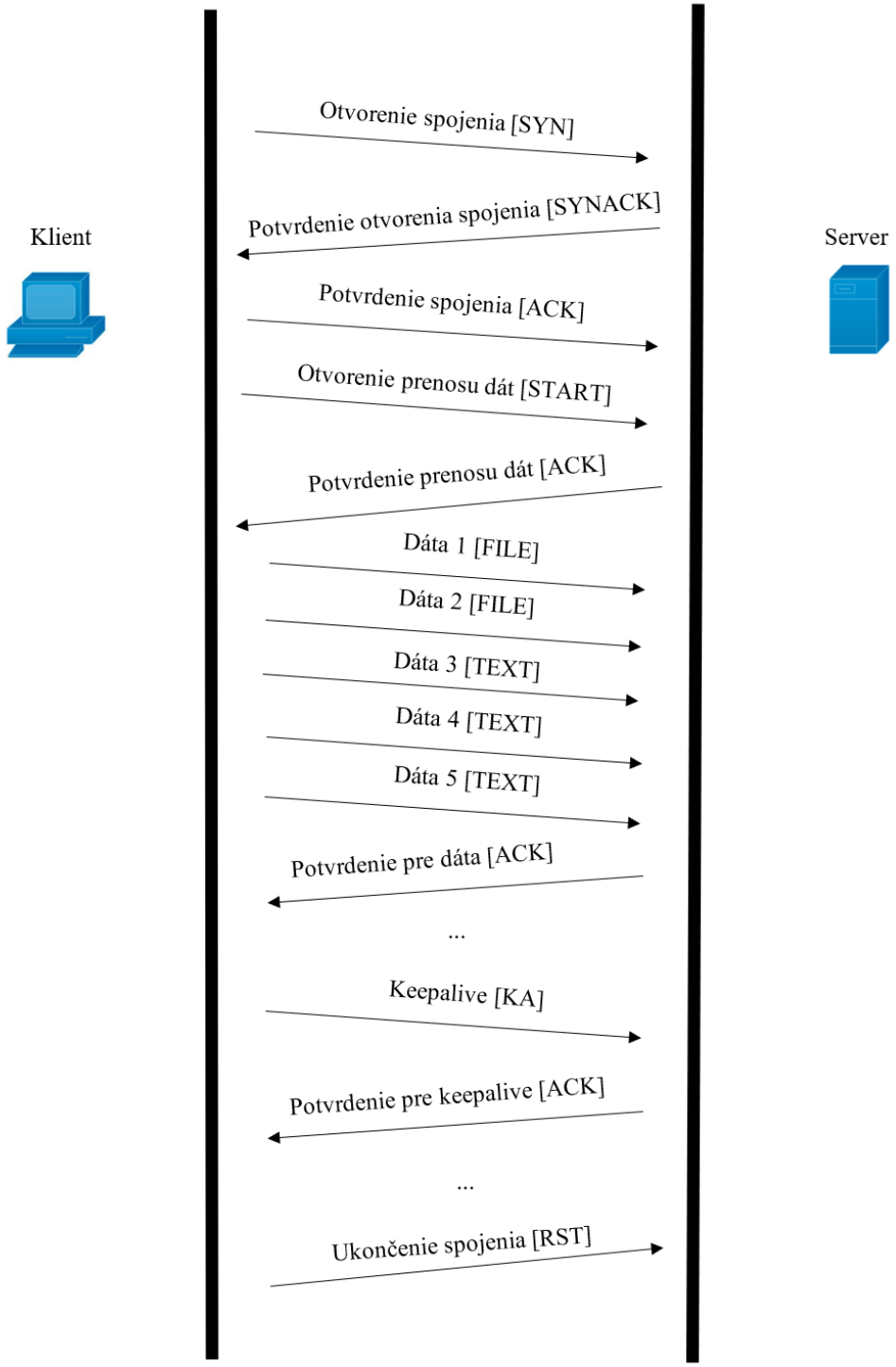
Server bude prijatie paketov od klienta potvrdzovať postupne (každý 10 paketov, prípadne iná hodnota) a klient na toto potvrdenie bude čakať.

Ak klient po odoslaní určitého množstva paketov od serveru obdrží ACK tak pošle ďalšiu várku paketov.

Ak klient po odoslaní várky paketov obdrží NACK tak túto várku paketov, kt bola na strane serveru označená ako chybná alebo nedoručená pošle znovu.

Takto sa zabezpečí, že server obdrží všetky pakety v použiteľnom a správnom stave.

Diagram spracovávania komunikácie (príklad)



Priebeh spojenia

1. Ak sa klient pripája na server, pošle správu s flagom SYN
2. Server na túto správu odpovedá správou SYNACK
3. Klient pošle serveru potvrdenie a spojenie je nadviazané
4. Následne si klient môže zvoliť ďalšiu akciu

a. Keepalive
b. Zaslanie textovej správy
c. Zaslanie súboru
d. Ukončenie spojenia (pošle sa správa s RST flagom)

Keepalive (metóda udržiavania spojenia)

Po úspešnom nadviazaní spojenia klienta so serverom bude v prostredí klienta na výber možnosť zapnutia alebo vypnutia keep alive teda udržiavanie spojenia.

Ak je u klienta aktívne keep alive, klienta bude v stanovenom intervale serveru zasielať paket s flagom KA. Server po obdržaní paketu s flagom KA odošle klientovi ACK a teda potvrdí, že spojenie je aktívne. Na základe výmeny týchto dvoch správ sa spojenie považuje za aktívne nakoľko obe strany preukázali svoju prítomnosť a činnosť.

Ak klient po odoslaní KA 3x po sebe nedostane od serveru žiadne ACK, spojenie sa bude považovať za ukončené/mŕtve, nakoľko sa predpokladá, že server je neaktívny.

Ak server za určité časové obdobie neobdrží žiadne KA alebo nieje aktívny iný druh spojenia.

Ak je u klienta aktívne KA a zároveň bude chcieť klient odoslať dáta serveru. Počas spojenia pre posielanie dát sa keep alive vypne a po ukončení dátového prenosu a všetkých s tým súvisiacich náležitostí sa opätovne zapne.

Zaslanie textovej správy

Po otvorení spojenia mojim vlastným 3 way hand shakom má klient možnosť zastať na server textovú správu. Zaslanie prebieha tým, že klient napíše do terminálu správu, kt. chce poslať. Tá sa následne transformuje na bajt string, pošle sa správa s START flagom. Táto správa bude obsahovať aj počet paketov nesúcich dáta do kt. bude správa v prípade potreby rozdelená.

Po START, klient čaká na ACK a následne sa bude cyklicky posielať dáta a to nasledovne:

Po zaslaní každých 5 paketov bude klient čakať na ACK od serveru, že pakety dorazili v poriadku.

Nakoľko server vie koľko paketov má dostať nie je potrebné aby klient informoval o ukončení posielania dát.

Zaslanie súboru

Zasielanie súboru prebieha podobne ako zaslanie súboru. S tým rozdielom, že predtým ako sa začne posielať obsah súboru, kt. pakety nesú flag TEXT rovnako ako textová správa. Je ako prvé zaslaný názov súboru spolu s cestou. Pakety nesúce náležitosti súboru sú označené flagom FILE.

Ukončenie spojenia

Ukončiť spojenie vedia obe strany a to zaslaním správy nesúcej flag RST. Ukončenie je okamžité a nevyžaduje žiadne potvrdenie druhej strany.

Ukončiť spojenie sa môže aj neplánovane a to uplynutím časového limitu počas, kt. nie je aktívne žiadne spojenie. Vtedy spojenia padá a neposielajú sa žiadne správy.

Ostatná dokumentácia

Implementácia

Kód je rozdelený na akoby dve časti:

1. Server

- `mode_server()`

Vytvára socket pre server a následne ho binduje so zadanými údajmi.

- `server_site(server_socket, server_addr_tuple)`

Zabezpečuje 3 way hand shake na strane serveru. Čakanie na žiadosť o spojenie od clienta reprezentované správou s flagom SYN. Následne sa posiela správa SYNACK a čaká sa na prijatie správy od clienta s flagom ACK. Ak dané kroky prebehli správne tak je spojenie nadviazané.

```
# cakanie na ziadost o spojenie SYN od klienta
data, client_addr_tuple = server_socket.recvfrom(RECV_FROM)
data = packet_reconstruction(data, False)

# ak prisla ziadost o spojenie SYN
if data.flag == SYN:

    # server posle klientovy SYN ACK
    initialization_packet = Mypacket(SYN + ACK, 0, 0, 0, "")
    server_socket.sendto(initialization_packet.__bytes__(False), client_addr_tuple)

    # cakanie na potvrdenie spojenia ACK od klienta
    data, client_addr_tuple = server_socket.recvfrom(RECV_FROM)
    data = packet_reconstruction(data, False)

    # ak prislo ACK tak spojenie je active
    if data.flag == ACK:
        print(f"Server: established connection with: {client_addr_tuple[0]}, port: {client_addr_tuple[1]}")
        server_as_receiver(server_socket, client_addr_tuple)
```

- `server_as_receiver(server_socket, client_addr_tuple)`

reaguje na prijaté flags: RST, KA, START.

V prípade, že príde START, čo je začiatok dátového prenosu pošle ACK a následne po chunkoch prijíma a pakety, kontroluje ich a ak je v chunku chybný paket vyžiada chunk znova prostredníctvom odoslania správy NACK. Ak v chunku nebol chybný paket prejde sa na prijímanie ďalšieho chunku.

```
received_crc = data.crc
data.crc = 0
calculated_crc = crc16(data.__bytes__(True))

if received_crc != calculated_crc:
    broken_packets = True
    broken_packets_local = True
```

Ak boli prijaté všetky packety tak sa prejde na ich usporiadanie podľa ich poradových čísel a následne a vypíše obsah správy alebo vytvorí súbor.

2. Client

- `mode_client()`

Vytvára socket clienta a následne zasiela žiadosť o spojenie so serverom prostredníctvom odoslania správy s flagom SYN. Čaká na SYNACK od serveru a následne zašle ACK.

Rovnako tak spúšťa keep alive funkcionalitu.

- `client_site(client_socket, server_addr_tuple)`

Zabezpečuje menu akcií pre klienta. Ponúka možnosť ukončenia spojenia alebo výber dátového prenosu spomedzi textovej správy a súboru.

- `client_as_sender(client_socket, server_addr_tuple, type)`

Funkcia slúži na poslanie textovej správy alebo súboru z klienta na server.

```
file_path_arr = [file_path[i:i+max_packet_data_size] for i in range(0, len(file_path), max_packet_data_size)]
arr_mess = [message[i:i+max_packet_data_size] for i in range(0, len(message), max_packet_data_size)]
```

```
# vypocet kolko bude fragmentov
temp_all_packets_arr = file_path_arr + arr_mess
num_of_packets_total = len(temp_all_packets_arr)
```

Dáta kt. sa majú poslať sú rozdelené na fragmetny podľa zadanej veľkosti.

Následne sú podľa množstva fragmentov fragmenty začlenené do chunkov a to rovnomerne a stým, že počet fragmentov v poslednom chunku môže byť menší.

```
num_of_chunks = math.trunc(num_of_packets_total / SIZE_OF_CHUNK)
size_of_last_chunk = num_of_packets_total % SIZE_OF_CHUNK
sizes_of_chunk_arr = [SIZE_OF_CHUNK] * num_of_chunks
sizes_of_chunk_arr.append(size_of_last_chunk)
```

Následne sa dané dáta zasielajú postupne po chunkoch s tým, že sa vždy čaká na odpoveď či bol daný chunk v poriadku a ak áno prejde sa na ďalší chunk ak bola chyba daný chunk sa odošle znovu.

Keep alive

`call_keep_alive(client_socket, server_addr_tuple)`

```
thread = threading.Thread(target=keep_alive, args=(client_socket, server_addr_tuple))
thread.daemon = True
thread.start()
```

`keep_alive(client_socket, server_addr_tuple)`

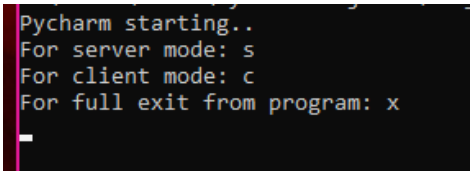
```
while True:
        if thread_status:
            ka_packet = Mypacket(KA, 0, 0, 0, "")
            client_socket.sendto(ka_packet.__bytes__(False), server_addr_tuple)

            data, address = client_socket.recvfrom(RECV_FROM)
            data = packet_reconstruction(data, False)

            if data.flag == ACK:
                print("Client keep_alive: connection is working..")
            else:
                print("Client keep_alive: connection ended")
```

```
                break
            time.sleep(KA_INTERVAL)
    pass
```

Používateľské rozhranie

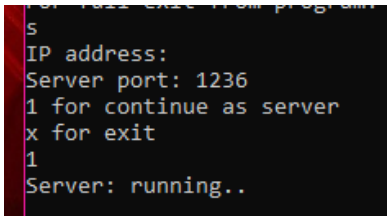
Program bude komunikovať s používateľom prostredníctvom konzolového rozhrania a textových výpiskov a rovnako bude z konzoly aj čítať vstupy zadané používateľom.

Úvodné menu



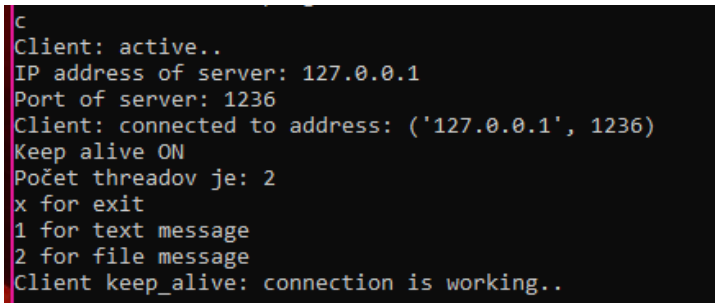
Po spustení programu sa vypíše hlavné úvodné menu kde si používateľ môže vybrať akú stranu chce spustiť (klient alebo server).

Server



Ak sme zvolili "s" pre server tak následne máme možnosť zadať IP adresu serveru a port na akom bude server počúvať. Následnej je ešte možnosť pokračovať ako server alebo možnosť exitu pre návrat do hlavného menu.

Client



Ak máme spustený server môžeme spustiť inštanciu pre klienta výber "c" v hlavnom menu.

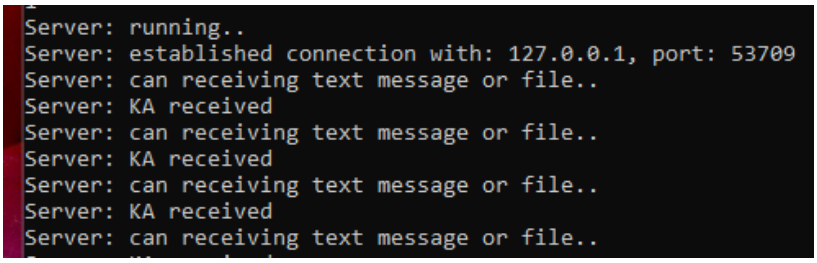
Následne na výzvu zadáme IP adresu serveru na kt sa chcem pripojiť a port na kt. server počúva.

Ak pripojenie prebehne úspešne dostaneme notifikácia o tom kam sme sa pripojili.

Ďalšia hláška nám oznamuje, že je aktívne keep alive a počet threadov.

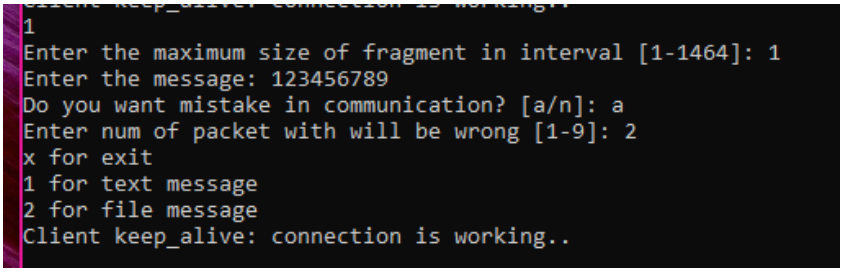
Ako posledné sa vypíše menu pre výber akú funkcionalitu klienta chcem zavolať.

Server po pripojení clienta



Ak sa na server pripojí klient, server toto pripojenie ohlási, následne sa vypisuje ak nám prichádzajú správy s keep alive správou.

Zaslanie textovej správy clientom

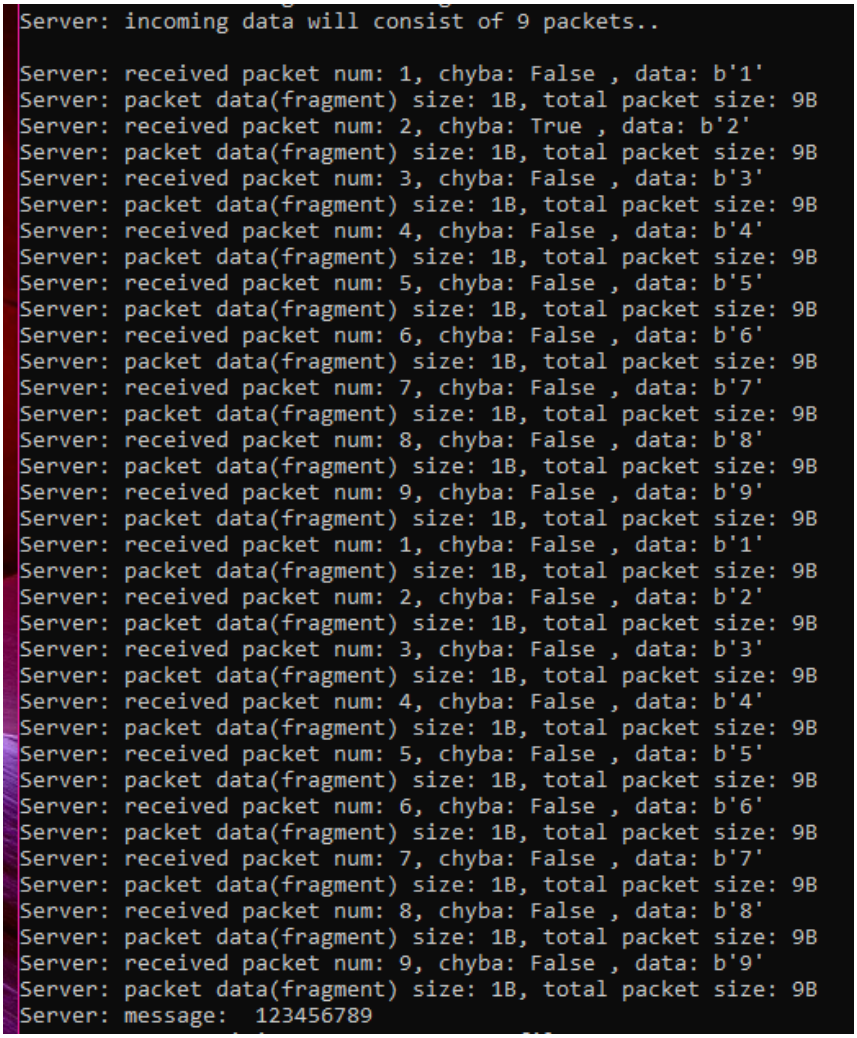


Ak sme v klientsom menu zvolili možnosť "1" teda zaslanie textovej správy máme možnosť zvoliť si veľkosť fragmentu. Následne dostaneme výzvu na zadanie správy, kt. chcem odoslať.

Nasleduje výber možnosti chyby kde v prípade, že chceme zadať chybu môžeme zvoliť z vypísaného intervalu číslo paketu kt. bude chybný.

Ak bolo odoslanie úspešné vidíme znova menu klienta. A zase budeme vidieť správy ohľadom toho či je spojenie aktívne.

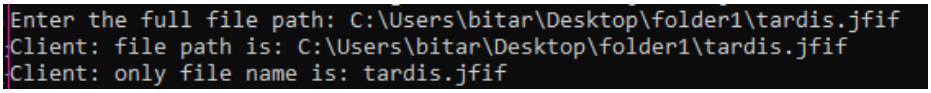
Prijatie textovej správy serverom



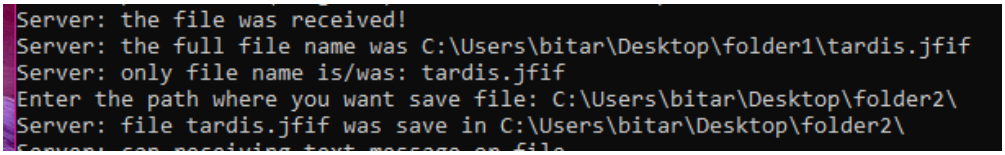
V prijatých paketoch bola chyba (vid. paket 2) a keďže veľkosť chunku je 10 tak preto sa to celé poslalo ešte raz.

Server vypíše koľko paketov čaká, následne vypisuje priebežne prijaté pakety a na záver vypíše znenie správy.

Posielanie súboru



Client



Server

Zasielanie súboru prebieha podobne ako posielanie správy s tým rozdielom, že na strane klienta nezadávame správu ale absolútnu cestu spolu s názvom súboru, kt. chcem odoslať a na strane serveru zadávame absolútnu cestu kam sa má súbor uložiť.

Splnenie funkcionalít

Nastavenie IP a port

Pri spúšťaní strany serveru rovnako ako aj pri klientovi má používateľ možnosť nastaviť IP adresu a port teda údaje pre komunikácie. Samozrejme na strane klienta je potrebné nastaviť port na, kt. počúva server a zadať adresu serveru inak komunikácia nebude možná.

Zvolenie maximálnej veľkosti fragmentu

Pri odosielaní správy zo strany klienta dostáva používateľ možnosť nastaviť maximálnu možnú veľkosť fragmentu, samozrejme je potrebné ak používateľ nastavil veľkosť z požadovaného intervalu teda 1-1464B.

Posielanie textových správ

Klient má možnosť vybrať poslanie textovej správy, následne túto správu napíše do konzoly a správa sa odošle serveru, kt. správu prijme, spracuje a do konzoly vo svojej inštancii programu vypíše obsah správy.

Prenosu súboru menšieho ako nastavená veľkosť fragmentu

V prípade, že posielame súbor, kt je menší ako nastavená veľkosť tak sa súbor normálne odošle ako aj pri inom prípade kedy by veľkosť súboru bola väčšia ako veľkosť fragmentu

Simulácia chyby pri prenose

Pri odosielaní správy alebo súboru je používateľovi na strane klienta kladená otázka či chceš do prenosu zaradiť aj chybu. Ak používateľ zvolí možnosť pre chybu následne je vyzvaný aby uviedol číslo paketu, kt. sa pošle ako chybný. Daný paket sa následne pri odoslaní odošle tak, že po výpočte crc mu je crc zmenená a teda druhá strana, server, pri kontrole tento paket vyhodnotí ako chybný.

Prenos 2MB súboru

Program dokáže poslať 2MB súbor bez toho aby to zabralo nejaký dlhší čas.

Testoval som aj súbor, kt. veľkosť bola cez 3MB a nebol žiadny problém.

Udržiavanie spojenia

Udržiavanie spojenia je zabezpečené pomocou KA správ, kt. posiela strana klienta.

Iné

Obe komunikujúce strany musia byť schopné zobrazovať:

- názov a absolútnu cestu k súboru na danom uzle,
- veľkosť a počet fragmentov.

Prijímajúca strana musí byť schopná oznámiť odosielateľovi správne aj nesprávne doručenie fragmentov. Pri nesprávnom doručení fragmentu vyžiada znovu poslať poškodené dáta.

Záver

Konštatujem, že program funguje správne a spĺňa všetky požadované funkcionality.

Zadanie bolo pomerne zaujímavé. Kombinuje prvky zo znalostí o sieťach a programovaní čo je pomerne zaujímavá výzva. Kód som sa snažil zostaviť tak aby bol prehľadný a jednoducho rozšíriteľný a som s jeho finálnou verziou spokojný.

Zaujímavé je aj dodať, že program by s dal donekonečna zlepšovať a rozširovať. Pridávať ďalšie funkcionality a rozširovať výpis informácií ohľadom priebehu spojenia.

Zmeny oproti návrhu

zmeny oproti návrhu nastali v nasledovných veciach:

- Zmena použitej implementácie pre výpočet CRC
- V návrhu uvádzam, že "Ak klient po odoslaní KA 3x po sebe nedostane od serveru žiadne ACK, spojenie sa bude považovať za ukončené/mŕtve, nakoľko sa predpokladá, že server je neaktívny." túto funkcionalitu som však nakoniec neimplementoval explicitným počítaním koľko KA sa neobdržalo ale vyriešil tom to čisto iba timerom teda ak sa 60s klient neohlási serve považuje klienta za mŕtveho a spojenie bude zrušené
- Ďalšia zmena je v hlavičke kedy sa atribút size v programe vôbec nevyužíva. Rozhodol som sa ho však nevymazať z dôvodu možnosti rozširovať zadanie alebo doimplementovania ďalších funkcionalít.

Technické špecifikácie prostredia

Program bude implementovaný v programovacom jazyku Python ver. 3.9 s kt. sa bude pracovať vo vývojovom prostredí PyCharm.

Program je funkčný za predpokladu dodržania všetkých špecifikačných potrieb pre spustenie python kódu (nainštalovaný jazyk python, dostupnosť knižníc..) a samozrejme prítomnosť všetkých potrebných externých súborov.