Anastasiia-Solomiia Hrytsyna (ID: 105323)

Implementácia vlastného protokolu

❖ Riešený problém (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).

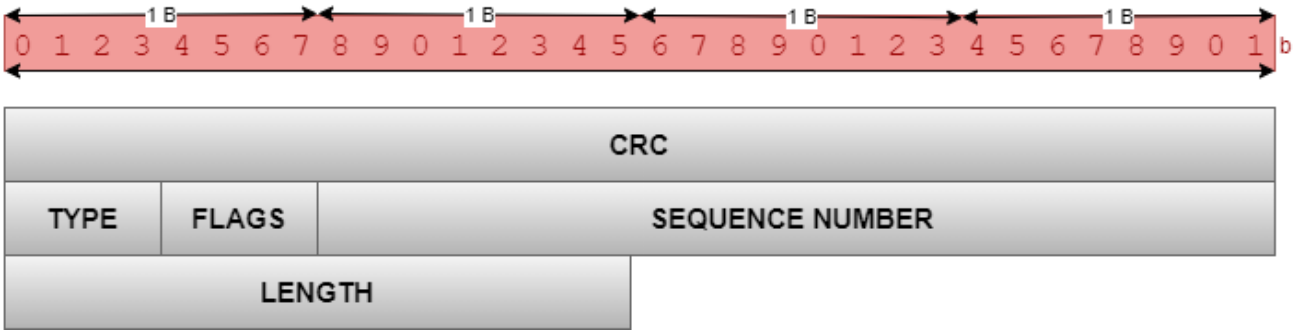
❖ Opis riešenia

▪ Implementačné prostredie

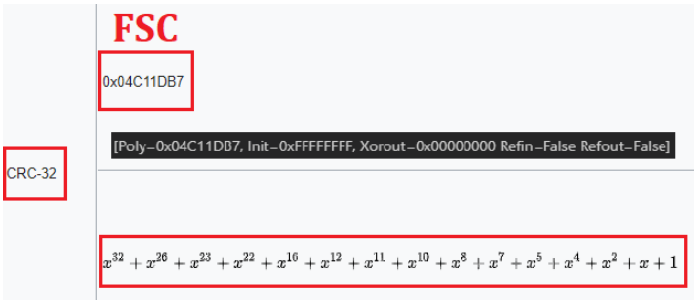
Ako už som to písala v návrhu projektu - zadanie ja som riešila v programovacom jazyku Python 3.9.0 pomocou prostredia Pycharm 2020.2.3. Naviac pre efektívnu implementáciu som použila ďalšie (väčšinou defaultne vbudované) knižnice:
- random (na generovanie seed-u pre časovače)
- os (na overenie directory a samotného súboru)
- socket (na spojenie dvoch používateľov a prenos dát medzi nimi)
- struct (na prácu s jednotlivými bajtmi)
- math (na matematické operácie s číslami)
- sys (na stopovanie vlákien)
- time (na vyrátanie trvalosti posielania dát alebo checkovanie keep-alive paketov)
- libscrc (na vyrátanie CRC) https://pypi.org/project/libscrc/
- threading (na spúšťanie niekoľkých funkcií súčasne)

▪ Header

V porovnaní s návrhom svoju vytvorenú hlavičku som takmer nevymenila (iba som pridala niekoľko nových TYPE-ov) a jej veľkosť sa zostala taká istá (10B). Teda najväčšia možná veľkosť jednotlivého fragmentu bude 1500B (dáta v Ethernete) - 20B (hlavička vnoreného IP protokolu) - 8B (hlavička vnoreného UDP protokolu) - 10B (veľkosť mojej hlavičky) = 1462B. Jednotlivé polia hlavičky sú také:



- Kontrolu správneho prenášania dát (paketov) zabezpečuje CRC (cyclic redundancy check). Ja som sa rozhodla použiť klasický CRC_32 s generujúcim polynom (viď obrázok)---
On je dôveryhodnejší ako CRC_16 a preto lepšie overuje dáta.



https://en.wikipedia.org/wiki/Cyclic_redundancy_check

Vstupom do CRC je postupnosť bitov. Do tej postupnosti nakoniec pripisujeme toľko núl, koľko ukazuje najvyšší exponent (v našom prípade pripočítame 32 nuly).
Generujúci polynóm ukazuje kde v deliteli budú sa nachádzať jednotky. Čiže tam, kde chyba x^(nejaké miesto) bude sa nachádzať nula. V našom prípade polynóm je 0b00000100110000010001110110110111
Teraz aby dostať CRC z pôvodnej postupnosti bitov musíme vydeliť ju našim polynómom (teda postupne zľava doprava urobiť logickú operáciu XOR). Zvyšok po delení (čiže 4-bajtove číslo čo bolo pred tým, ako sme dostali 0 v delení) je nás výsledok – CRC, krotí už sa nemení (lebo XOROUT = 0).

- TYPE určuje typ poslaného paketu. Celkovo potrebujem iba 4 bity naňho a spolu s FLAGS mne to dá 1 bajt aby sa to dokopy zarovnalo. Sú 7 rôznych druhov typov:
  1111 (CONNECT) – paket pre nadviazanie spojenia
  0000 (DISCONNECT) – paket pre ukončenie spojenia
  1000 (INIT) - inicializačný paket pre posielania informácie o tom, čo chceme poslať
  0001(LAST) - posledný paket pre ukončenie spojenia
  1001 (KEEP_ALIVE) - keep_alive paket pre udržiavanie spojenia
  0101(TXT) - txt paket (správa)
  1010 (FILE) - nejaký súbor (file)
- FLAG určuje či posielanie dát prešlo správne a či sme nedostali žiadnu chybu (či fragment nebol poškodený). Sú 3 možnosti nastavenia príznakov:
  1111 (ACK) - prenos paketu prebehol úspešne a on je celý
  1001 (NACK) - prenos paketu je neúspešný, žiadam o preposielanie ešte ráz
  0000 - defaultny flag
- SEQUENCE NUMBER určuje nám poradové číslo prenášaného fragmentu. Pri najmenšej možnej veľkosti fragmentu - 1 bajt, je pri 3-bajtovom sequence number $2^{24} \sim 16{,}8$ mil možností, čo znamená, že dokážeme poslať 16MB súbor, a to je viac ako postačujúce pre naše zadanie.
- LENGTH je iba dĺžka našich prenášaných dát (čiže v maximálnej možnosti - 1462B).

- **Princip komunikácie**

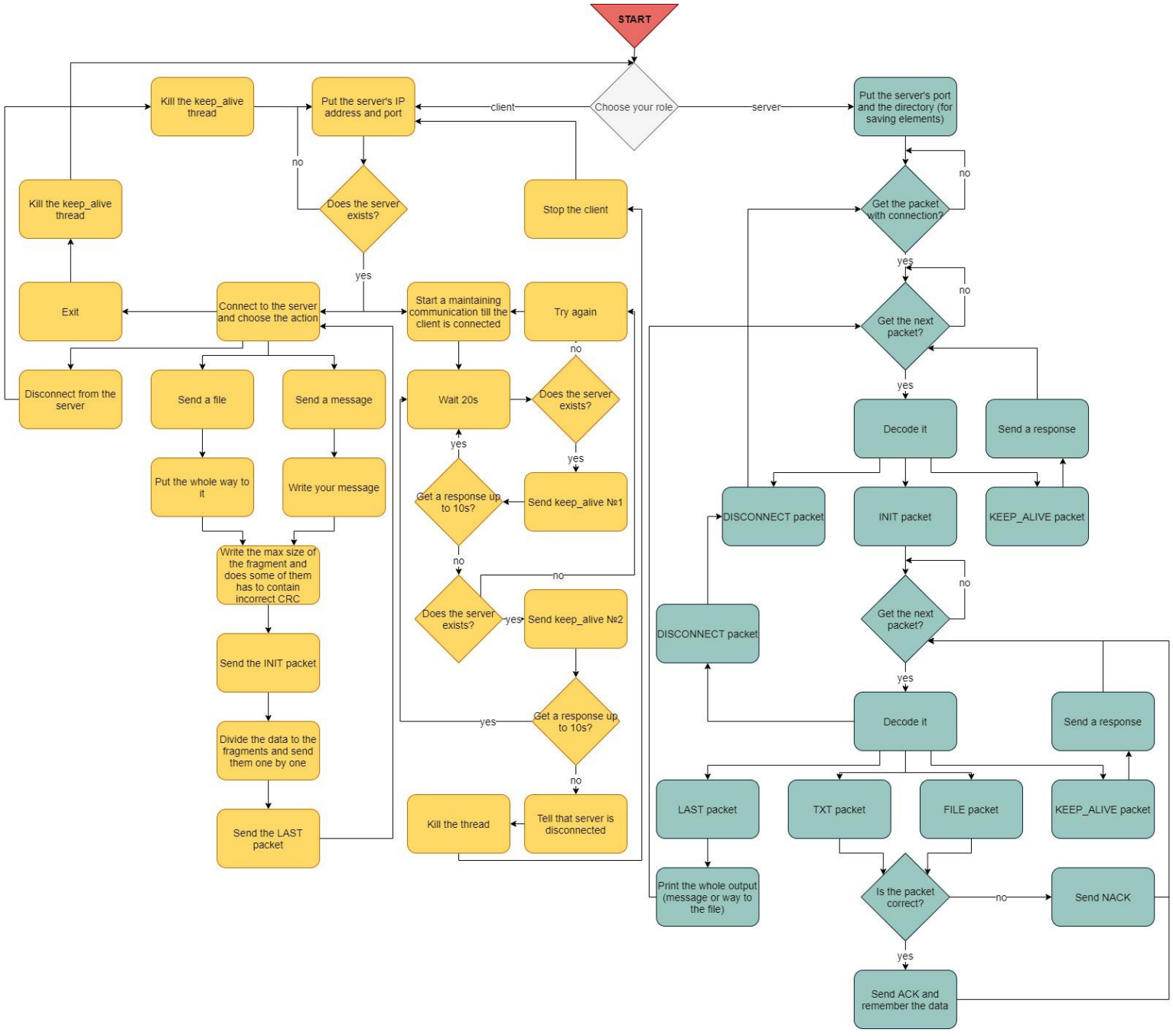
- Posielanie dát

V podstate, v porovnaní s návrhom projektu ja som neurobila nejaké veľké zmeny v celkovej komunikácie (iba niekoľko nevýznamných oprav). Hlavnú myšlienku viď na diagrame (krabička 'send data' v klientovi bude vysvetlená nižšie).
1. Serveru som pridala možnosť skončiť hocikedy počas behu programu.
2. Pridala som ešte jeden paket pre spojenie s serverom. Teda, ešte pred posielaním INIT paketu ja overujem či sa dá pripojiť sa na server a budovať komunikáciu medzi uznali.
3. Hneď po potvrdení pripojenia na server ja spustím ešte jeden socket, ktorý bude bežať paralelne s klientom. On bude udržiavať spojenie medzi klientom a serverom, čiže keď aspoň jeden z nich zanikne, tak sa zabije aj to vlákno s socketom na podporovanie vzťahu. Samotný algoritmus udržiavania spojenia je popísaná nižšie.
4. Hlavný princíp komunikácie sa zostal ten istý. Vysielam dáta pomocou selective repeat ARQ metódy (vysvetlenie funkčnosti je v návrhu - všetko platí tak isto). Zo strany servera udržiavanie spojenia ja už som vysvetlila vyššie (v bode 6). Avšak klient to ma trochu zložitejšie. V klientovi bude ešte jeden socket, ktorý bude bežať v paralelnom vlákne. Takže počas celého spojenia (aj počas posielania dát) budú sa posielať keep_alive-y dvoch typov. Prvý bude sa vysielať pravidelné každých 20s. Ak dostane odpoveď - začne od začiatku. Ak náhodou do 10s na neho nepríde odpoveď zo servera, tak sa hneď pošle keep_alive druhého typu. Opäť čakáme 10s a ak ani dovtedy server nepošle odpoveď, tak vyhlásim, že server je disconnected a stopnem aj naše vlákno, aj používateľa. Inak zopakujem opäť všetko od začiatku. Naviac som pridala to, že preposielame paket keď dostaneme NACK odpoveď od servera alebo keď už príliš dlho nedostaneme vôbec žiadny response (u mňa je ten čas nastavený na 0.2s).

5. Ešte by som ešte rada upresnila - server taktiež ma svoje okno. Čiže on očakáva na nejaký konkrétny paket (aby dáta sa zapisovali postupne). Ak príde akurát ten paket, ktorý sa očakáva, tak ho zapíše do final verzie, inak pridá ho do bufferu (okna) a vyberie ho vtedy, keď ho bude potrebovať (keď on bude ďalší v postupnosti).
6. A posledné - keď na server už bol pripojení nejaký klient, ale server nedostane žiadny paket počas 35s (ani keep_alive, ani niečo iné), tak vyhlási, že client sa odpojil a skončí.

- Udržiavanie spojenia

Zo strany servera udržiavanie spojenia ja už som vysvetlila vyššie (v bode 6). Avšak klient to ma trochu zložitejšie. Aby sme vedeli, či server je v sieti, musíme pravidelne kontrolovať spojenie. Za to je zodpovedný keep_alive rámec (TYPE = 1001). Princíp jeho fungovania je ďalší: v klientovi bude ešte jeden socket, ktorý bude bežať v paralelnom vlákne. Takže počas celého spojenia (aj počas posielania dát) budú sa posielať keep_alive-y dvoch typov. Prvý bude sa vysielať pravidelné každých 20s. Ak dostane odpoveď - začne od začiatku. Ak náhodou do 10s na neho nepríde odpoveď zo servera, tak sa hneď pošle keep_alive druhého typu. Opäť čakáme 10s a ak ani dovtedy server nepošle odpoveď, tak vyhlásim, že server je disconnected a stopnem aj naše vlákno, aj používateľa. Inak zopakujem opäť to iste od začiatku.



❖ Zhodnotenie

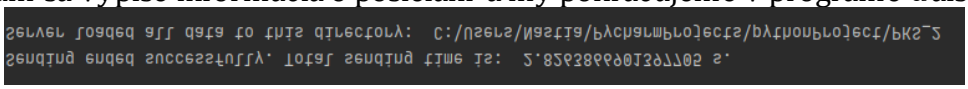
Ja si myslím, že môj projekt je úspešný, lebo spĺňa všetky minimálne požiadavky a funguje korektné. Implementovala som zložitejšie metódy pre posielanie a udržiavanie spojenia, čo dosť vylepšilo a zrýchlilo posielanie informácie.
Samozrejme, že je ešte obrovský priestor na vylepšovanie. Napríklad rada by som pridala ešte jeden návrh, ktorý mi napadol neskôr (a preto som ho nestihla implementovať), avšak pre vylepšenie programu by bol užitočný. Keďže klient posiela iba tie pakety, ktoré sa nachádzajú v jeho okne, tak číslovať ich kľudne môžeme od 0 až 2*(veľkosť okna). Čiže keby sme si zvolili veľkosť okna 128 paketov (inak v programe ja mám iba 8), tak číslovať pakety by sme mohli iba do 256 a potom pridávať nejaké offsety na výpis v serveri. Tým pádom sequence number by sa zmestilo aj do 1B, čo by nám ešte viac ušetrilo miesto pre dáta.
Taktiež som nestihla pridať nejaké GUI do programu, preto všetko beží cez konzolu, avšak aj tam hlavná funkčnosť je zachránená. Aby nejak lepšie to bolo znázornené, tak som aspoň urobila dissector na Wireshark, ktorý "pekne" zachytáva komunikáciu na 3333 porte. Napríklad aby poslať približne 2MB súbor vstup bude vyzerať nejak takto:

```
> (1) Server
> (2) Client
2
Write IP address of the server:   192.168.0.108
Write port of the server:   3333
Client (IP address: 192.168.0.108 Port: 65438) is connected to the server.

> (1) Send some text message.
> (2) Send a file.
> (3) Disconnect.
> (0) Exit.
2
Write the way to the file:   C:\Users\Nastia\Desktop\Stuff\Христя_фото\20200816_201118.jpg
Put the fragment's size (1-1462):   1462
Print 0 if you don't want to have incorrect packet or 1 to mess up the CRC in one packet.1
Sending keep-alive №1 to the server.

Server was informed about sending.
The total size of the data is  2163465 B. The data will be divided into  1480 pieces.
```

Na konce nám sa vypíše informácia o posielaní a my pokračujeme v programe ďalšie...



Wireshark zachytí napríklad táketo...