**FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA**

Ilkovičova 2, 842 16 Bratislava 4

**2022/2023**

Počítačové a komunikačné siete

**Zadanie č.2**

**Komunikácia s využitím UDP protokolu**

**Cvičiaci: Ing. Miroslav Bahleda, PhD. Vypracoval: Šimon Valíček**

**Čas cvičení: Piatok 8:00 – 9:40 AIS ID: 116320**

Obsah

**Návrh programu a komunikačného protokoli**

1. **Úvod........................................................................................................................................3**
   1. *Opis zadania......................................................................................................................3*
   2. *Voľba implementačného prostredia..................................................................................3*
2. **Návrh hlavičky..........................................................................................................................3**
3. **ARQ..........................................................................................................................................6**
4. **Checksum.................................................................................................................................7**
5. **Sekvenčný diagram a metódy na udržanie spojenia..................................................................7**
6. **Zmeny v porovnaní s návrhom.................................................................................................10**
7. **Úvod**
   1. **Opis zadania**

**Názov:** Komunikácia s využitím UDP protokolu

**Zadanie úlohy:**

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 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.

* 1. **Implementačné prostredie**

Na implementáciu zadania som si zvolil jazyk Python, konkrétne verziu 3.10.8 a implementačné prostredie VisualStudio Code. V implementácii môjho programu som využil tieto externé knižnice:

Obrázok, na ktorom je text

Automaticky generovaný popis

1. **Návrh hlavičky**

Na úvod treba uviesť, že veľkosť packetu má byť 1500B. Nakoľko sa jedná o návrh protokolu nad protokolom UDP, jednotlivé Byty sú uvedené nasledovne. Prvých 20 Bytov je rezervovaných pre IPv4 protocol a následne 8Bytov zaberá UDP protocol. Zostalo nám tak 1472 Bytov, s ktorými budeme pracovať. Z týchto zvyšných 1472 Bytov potrebujem vytvoriť vlastnú hlavičku, dátovú časť a nejakú časť pre Checksum, kde bude prebiehať overovanie, či dáta prišli na cieľovú adresu nepoškodené. Môj program obsahuje dva typy hlavičiek- Rozlišuje hlavičku pre systémové správy a hlavičku pre textové a súborové správy.

Obe tieto hlavičky majú spoločný prvý Byt, ktorý reprezentuje, o aký typ správy sa jedná. Ako som už spomínal vyššie, typov správy budem rozlišovať tri a to konkrétne:

**m -** bude znamenať, že sa jedná o textovú správu

**f -** bude znamenať, že sa jedná o súbor

**s -** bude znamenať, že sa jedná o systémovú správu (tie sú generované systémom automaticky)

Tieto tri písmenká budú reprezentované ich hodnotou v ascii tabuľke.

**V prípade, že sa jedná o textovú alebo súborovú správu bude zvyšok hlavičky uložený v 9 Bytoch, ktoré sú rozdelené nasledovne:**

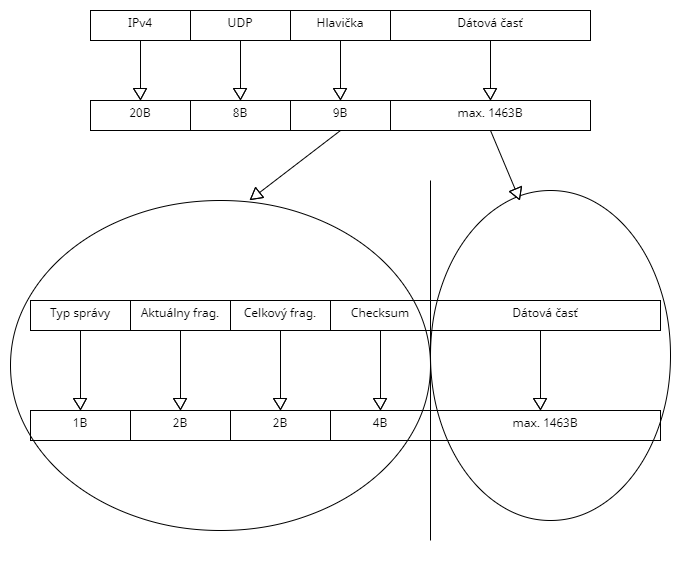
**Byty 2-3** budú predstavovať číslo aktuálneho fragmentu správy, ktorý sa posiela.

**Byty 4-5** budú predstavovať celkový počet fragmentov, na ktoré je správa rozdelená.

**Byty 6-9** sú rezervované pre checksum

Po odčítaní 9 Bytov pre hlavičku z pôvodných 1472 Bytov, nám zostalo dokopy 1463 Bytov.

To je konečný počet, ktorý predstavuje veľkosť v Bytoch pre dátovú časť. Dátová časť mojej správy (respektíve jej obsah) môže predstavovať maximálnu velkosť 1463 Bytov. V opačnom prípade musíme tento obsah rozdeliť na viacero fragmentov a posielať ho po častiach. Reprezentácia textového alebo súborového packetu vyzerá v mojom programe tak, ako je zobrazené na diagrame nižšie.

****

**V prípade, že sa jedná o systémovú správu, bude v hlavičke okrem Bytu reprezentujúceho typ správy už len jeden ďalší Byt. Ten v sebe nesie číslo, ktoré reprezentuje o aký druh správy sa jedná. Jednotlivé čísla sú tomuto Bytu pridelené nasledovne:**

**1** – reprezentuje ukončenie posielania textovej správy

**2** – reprezentuje ukončenie posielania súboru

**3** – reprezentuje inicializáciu komunikácie posielania súboru

**5** – hovorí o tom, že sa jedná o správu „keep alive“ pre udržiavanie spojenia

**6** – reprezentuje zahájenie spojenia medzi vysielačom a prijímačom

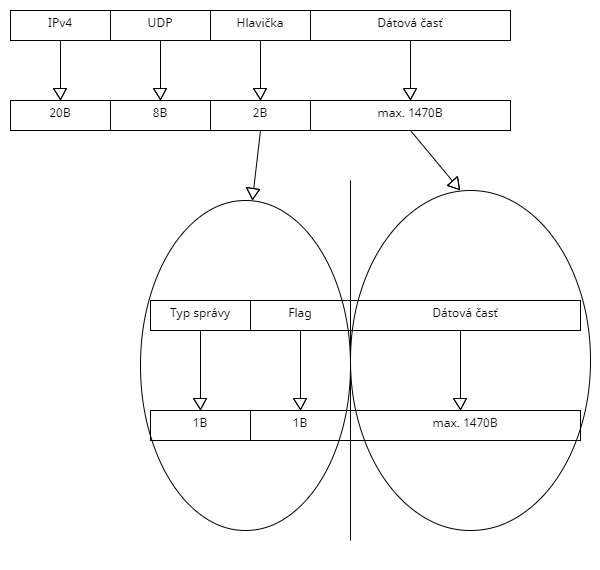
**7** – prijímajúca strana ju posiela vtedy, ak úspešne obrdžala fragment textovej správy, ktorý je skontrolovaný pomocou checksum

**8** – prijímajúca strana ju posiela vtedy, ak úspešne obdržala fragment súboru, ktorý je kontrolovaný pomocou checksum

**9** – správa reprezentujúca error, prijímajúca strana ju posiela, ak fragment súboru alebo textovej správy prišiel s chybou

**10** – reprezentuje obdržanie správy o zahájení posielania súboru, je posielaná prijímajúcou stranou

Pri systémových správach vyzerá hlavička a telo správy nasledovne.



1. **ARQ**

Pre implementáciu môjho programu som sa rozhodol využiť metódu **ARQ – Stop & wait**.

ARQ (automatic repeat request) je metóda prenosu informácii medzi dvoma uzlami. Zabezpečuje, že sa informácie nestratia kvôli zahodeným dátovým blokom, a zároveň že budú doručené v správnom poradí. ARQ – stop & wait funguje na nasledujúcom princípe. Vysielač pošle prijímaču správu, a ten na základe checksum vyhodnotí, či prišiel dátový blok v poriadku, alebo sa pri procese posielania poškodil. Ak je blok vyhodnotený správne, vyžiada si ďalší. Ak je blok vyhodnotený chybne, pošle žiadosť o znovuzaslanie chybného dátového bloku.

1. **Checksum**

Kontrolný súčet (Checksum) sa využíva na ošetrenie správnosti dátového bloku. Najprv sa na strane odosielateľa vypočíta hodnota checksum, ktorá sa následne po prenesení správy na stanu prijímateľa opäť vypočíta a porovná. Znamená to, že obe strany poznajú algoritmus na jej výpočet. V mojej implementácii riešenia budem využívať knižnicu libscrc, ktorá používa sumu CRC32. CRC32 – Cyclic Redundancy Checksum 32-bit zaberá veľkosť 32 bitov, teda 4 Bytov, ako je uvedené v mojom návrhu protokolu v kapitole 2. Funguje na princípe porovnaní dĺžok správ. V prípade že sa tieto dĺžky rovnajú, vieme s veľkou pravdepodobnosťou predpovedať, že pri prenose dát nedošlo ku chybe.

**Na príklade nižšie sa pokúsim podrobnejšie vysvetliť princíp používania konkrétnej kontrolnej sumy CIRC32.**

Funguje v podstate na princípe delenia polýnomov polynómmi, ktoré sú reprezentované bitmi. Delenie prebieha pomocou XOR.

Povedzme, že máme nasledujúcich 12 bitov - ***0110 1100 1101*** , tie reprezentujú náš delenec. Náš deliteľ bude reprezentovaný nasledovne – **1001.** Vydelíme teraz nášho nášho delenca deliteľom, akoby to boli polynómy. Berieme prvé 4 čísla z delenca a aplikujeme naňho a na nášho deliteľa metódu XOR. Ak je prvé číslo v riadku s výsledkom nula, posúvame sa o číslo v delencovi doprava. Takto pokračujeme, až kým nezískame chcený výsledok.

1. **Sekvenčný diagram a metóda udržiavania spojenia**

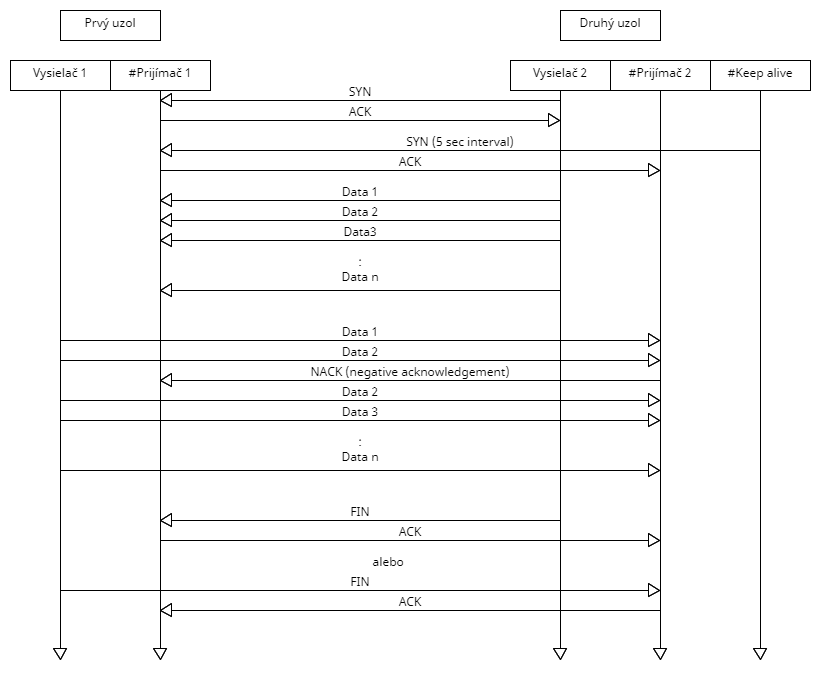
Na udržanie spojenia mi slúži metóda „Keep alive“. Tá beží na samostatnom threade na jednom z uzlov. Pomocou tejto metódy posiela uzol systémovú správu druhému uzlu v intervale 5 sekúnd. Ten mu odpovedá, akonáhle správu obrdží. Ak nastane z nejakého dôvodu odpojenie prijímača, vysielač posiela správu dookola na hluchý port, avšak bez odpovede. Ak nastane odpojenie vysielača, prijímač neprijíma žiadne „keep alive“ správy, a teda na ne ani neodpovedá.

V mojej implementácii je komunikácia medzi dvomi zariadeniami reprezentovaná nasledovne:

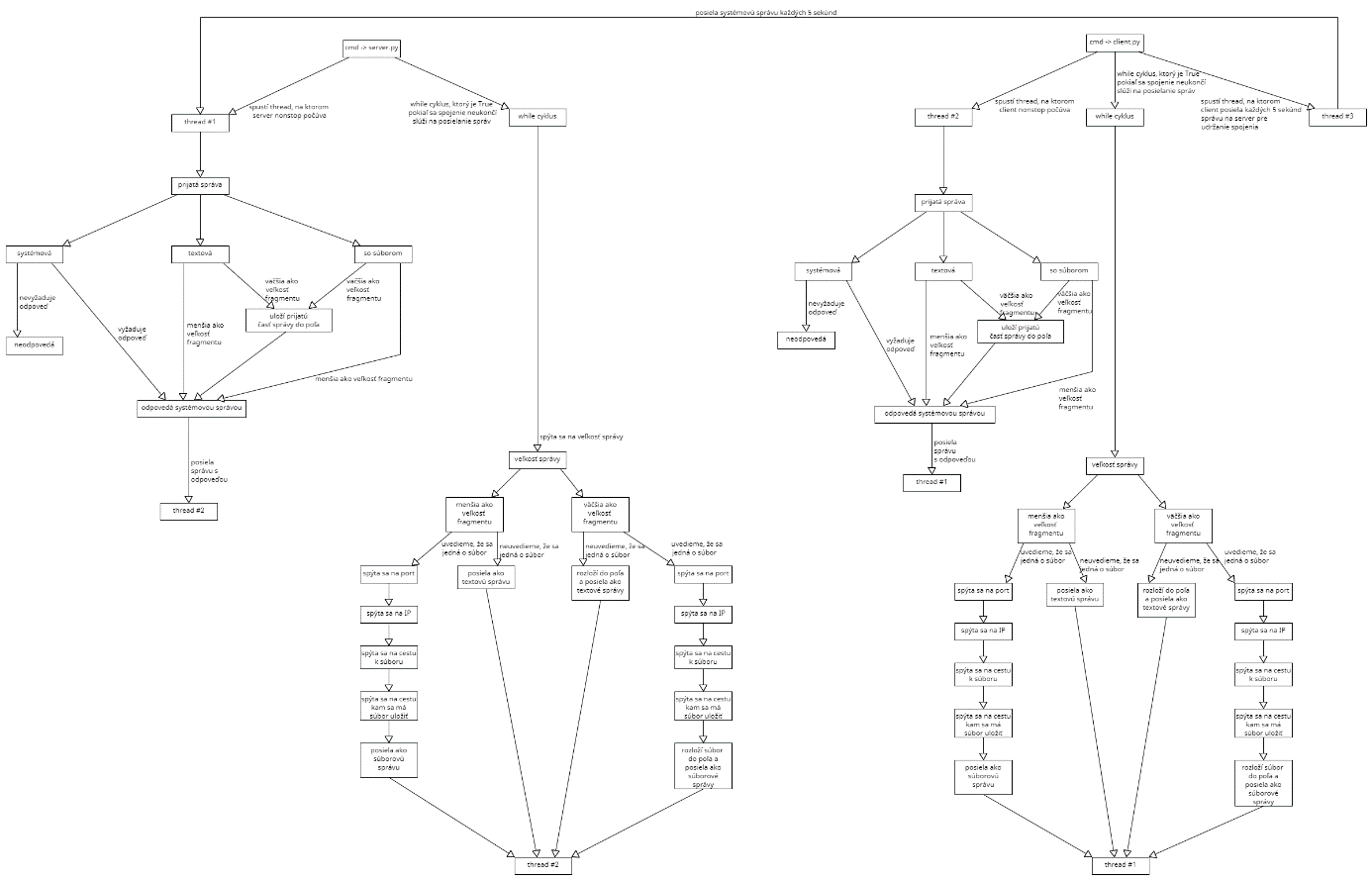
* Obe zariadenia sú vysielačom aj prijímačom súčasne
* Vzhľadom k predošlému bodu vyžaduje moja komunikáciu tri thready.
* Na jednej strane komunikácie je uzol, ktorý pomocou threadu neustále počúva a prijíma správy. Pomocou for cyklu, ktorý je vždy pravdivý (pokiaľ nie je určené inak) posiela správy druhému uzlu.
* Na druhej strane komunikácie je uzol, ktorý taktiež pomocou threadu nepretržite počúva a prijíma správy. Rovnako ako prvý uzol, aj tento využíva pravdivý for cyklus na posielanie správ (opäť, pokiaľ nie je určené inak). Tento druhý uzol zároveň využíva ďalší thread, pomocou ktorého v intervale 5 sekúnd odosiela správy prvému uzlu na udržanie spojenia. Ten na ne odpovedá.

**Zhrnutie:** Môj program teda využíva dokopy tri thready, pričom oba komunikačné uzly majú zároveň rolu prijímača aj vysielača.

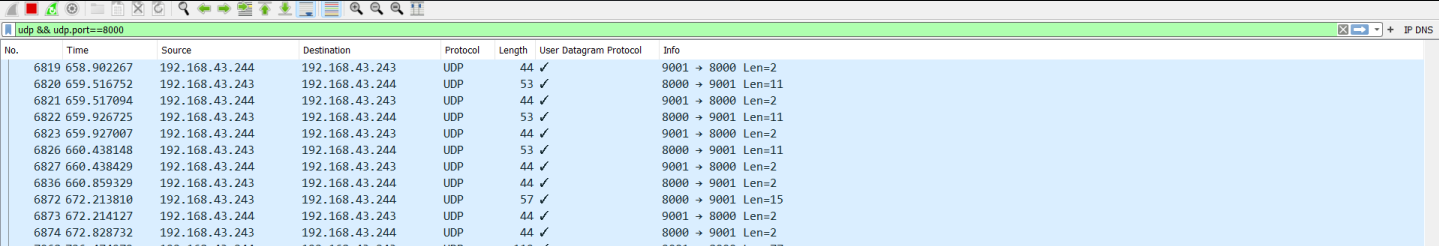
Pre lepšie vysvetlenie uvádzam sekvenčný diagram môjho programu.



Pre ešte lepšie vysvetlenie uvádzam úplný flowchart môjho programu s podrobnejším popisom jednotlivých jeho krokov. (Kvôli vysokému rozlíšeniu prikladám ešte tento diagram samostatne v prílohe)-



Pre ešte lepšie pochopenie prikladám screenshot z Wiresharku.



Na obrázku je zachytená časť textovej komunikácie medzi dvoma zariadeniami. Na začiatku môžeme vidieť, ako jedno zariadenie posiela druhému správu rozdelenú na fragmenty o veľkosti 2, nakoľko hlavička o veľkosti 9 + veľkosť fragmentu 2 = 11B. Druhé zariadenie mu odpovedá systémovými správami, ktoré majú veľkosť 2B. Po ukončení tejto komunikácie pošle posielajúce zariadenie správu o ukončení správy, ktorá je o veľkosti 2B a prijímajúce zariadenie mu odpovie tiež 2B správou, že ju prijalo a správu začne skladať. Vzápätí toto zariadenie, ktoré bolo doposiaľ prijímajúce sa rozhodne poslať správu o veľkosti 15B (9B hlavička a 6B dátová časť správy) bez nutnosti prepínania, ako môžeme vidieť na obrázku, nakoľko obe zariadenia vedia počúvať aj prijímať súčasne. Následne prebehne výmena systémových správ o veľkosti 2B medzi týmito zariadeniami, ktorá značí koniec posielania správy.

1. **Zmeny v porovnaní s návrhom**

* Po konzultácii na cvičení som sa rozhodol pre redukciu hlavičky pri posielaní textových a súborových správ. V návrhu som uvádzal hlavičku, ktorá obsahovala tri krát znak „/“ pre oddelenie jednotlivých Bytov, čo je nezmysel, nakoľko každá správa sa posiela premenená na Byty a podľa poradia Bytu v správe si viem jednotlivé údaje oddeliť. Ďalej táto hlavička obsahovala 4 Bytové miesto pre číslo aktuálneho fragmentu a taktiež 4 Bytové miesto pre číslo celkového počtu fragmentov. Tuto som sa mylne domnieval, že budem čísla posielať reprezentované ako reťazce znakov, čo bola chybná domnienka a po konzultácii som zistil, že 2 Byty mi bohate stačia na posielanie čísla o veľkosti 2^16, čo je pre môj program viac než dostačujúce. Odstránil som tak 3 zbytočné Byty z hlavičky, ktoré niesli znak oddelovača a ďalších 8 Bytov pre čísla fragmentov som zredukoval do 4 Bytov. Ušetril som tak 7 Bytov z hlavičky, namiesto ktorých som dal 4 Bytový checksum, a teda stále mi zostali 3 Byty k dobru. Pôvodná 12 Bytová hlavička má teraz veľkosť 9 Bytov a obsahuje viac informácií, ako obsahovala hlavička uvedená v návrhu.
* Po konzultácií na cvičení som sa rozhodol zmeniť moju ARQ metódu. Pôvodne som plánoval využiť upravenú metódu ARQ Selective repeat, ktorá by mala nekonečný buffer a posielala by zoznam všetkých chybných packetov na konci. Vzhľadom na to, že môj program beží na viacerných threadoch, ktoré neustále počúvajú a for cykloch, ktoré zapisujú vždy, keď im je povedané, nezostávalo mi nič iné, než využiť metódu ARQ Stop & wait, nakoľko k jednotlivých chybným fragmentom by som sa pomocou for cyklu na konci nevedel vrátiť. Musím teda zastaviť cyklus vždy po odoslaní packetu a počkať na odpoveď či je potrebné packet opäť odoslať, alebo nie.
* Nasledujúcu, poslednú zmenu oproti môjmu návrhu som si dovolil urobiť samovoľne, nakoľko som vyhodnotil, že pri systémových správach, ktoré nefragmentujem je zbytočné, aby som v hlavičke držal údaje o poradí fragmentu, celkovom počte fragmentov, či kontrolnej sume, ktorá je pri rovnako dlhých správach zbytočná, nakoľko počíta s dĺžkou fragmentu. Z dôvodov vyššie uvedených som sa rozhododl pridať druhý typ hlavičky, ktorý budú niesť všetky systémové správy. Táto hlavička obsahuje iba 2 Byty, pričom prvý Byt reprezentuje typ správy a druhý reprezentuje akýsi flag, na základe ktorého môj program rozlišuje, o akú systémovú správu sa jedná.