Matej Delinčák, ID: 102895

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

UDP komunikátor

2020/21, ZS

Matej Delinčák

STU FIIT

Matej Delinčák, ID: 102895

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.

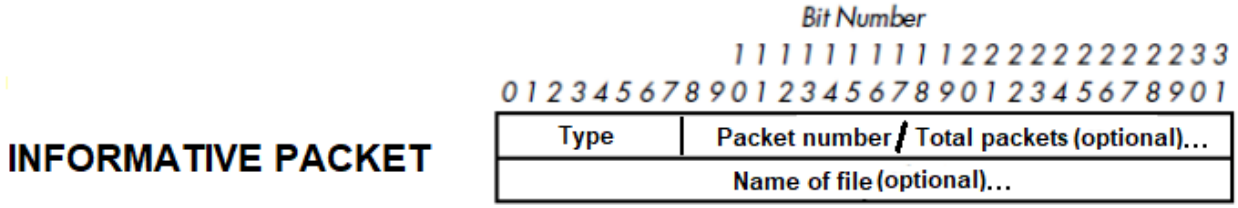
Program musí obsahovať kontrolu chýb pri komunikácii a znovuvyžiadanie chybných fragmentov, vrátane pozitívneho aj negatívneho potvrdenia. Po prenesení prvého súboru pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 10-60s pokiaľ používateľ neukončí spojenie. Odporúčame riešiť cez vlastne definované signalizačné správy.

Matej Delinčák, ID: 102895

Návrh protokolu

Vlastná hlavička

Môj protokol bude mať dve rozličné hlavičky. Informatívny paket a dátový paket. Informatívny paket bude vyzerať nasledovne:



Pole ***Type*** bude obsahovať flagy o akú správu ide a bude mať veľkost 1B:

Strana klient bude používať:

„0“ – začatie komunikácie a nepoužijem ostatné polia

„1“ – keep alive správa a nepoužijem ostatné polia

„2“ – bude poslaná správa a použijem pole ***Total packets***

„3“ – bude poslaný súbor a použijem obe polia ***Total packets*** ako aj ***Name of file***

Strana server bude používať:

„4“ – paket prišiel v poriadku a použijem len pole ***Packet Number***

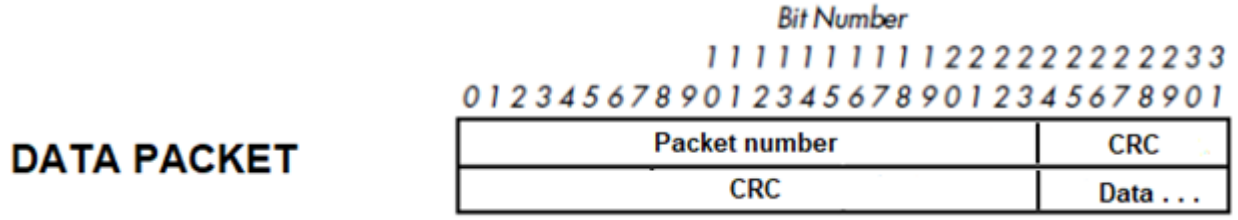
„5“ – paket nepríšiel v poriadku a použijem len pole ***Packet Number***

„6“ – všetky pakety sú v poriadku a nepoužijem ostatné polia

V poli ***PacketNumber/ Total packets*** bude počet paketov, alebo poradové číslo paketov, ktoré budú poslané. Použil som 3B, práve preto, lebo máme vedieť poslať 2MB súbor. To je približne 2 milióny bajtov (ak by veľkosť fragmentu bolo 1B). A v ***Name of file*** poli bude názov súboru.

Na obrázku som označil dve polia ako optional. To preto, lebo keď posielam začatie komunikácie a keep alive, tak ich nepotrebujem použiť. Ak posielam správu, použijem pole **Total packets,** ale nie ***Name of file***. A ak posielam súbor, tak použijem všetky tri.

Ďalej opíšem dátový paket:

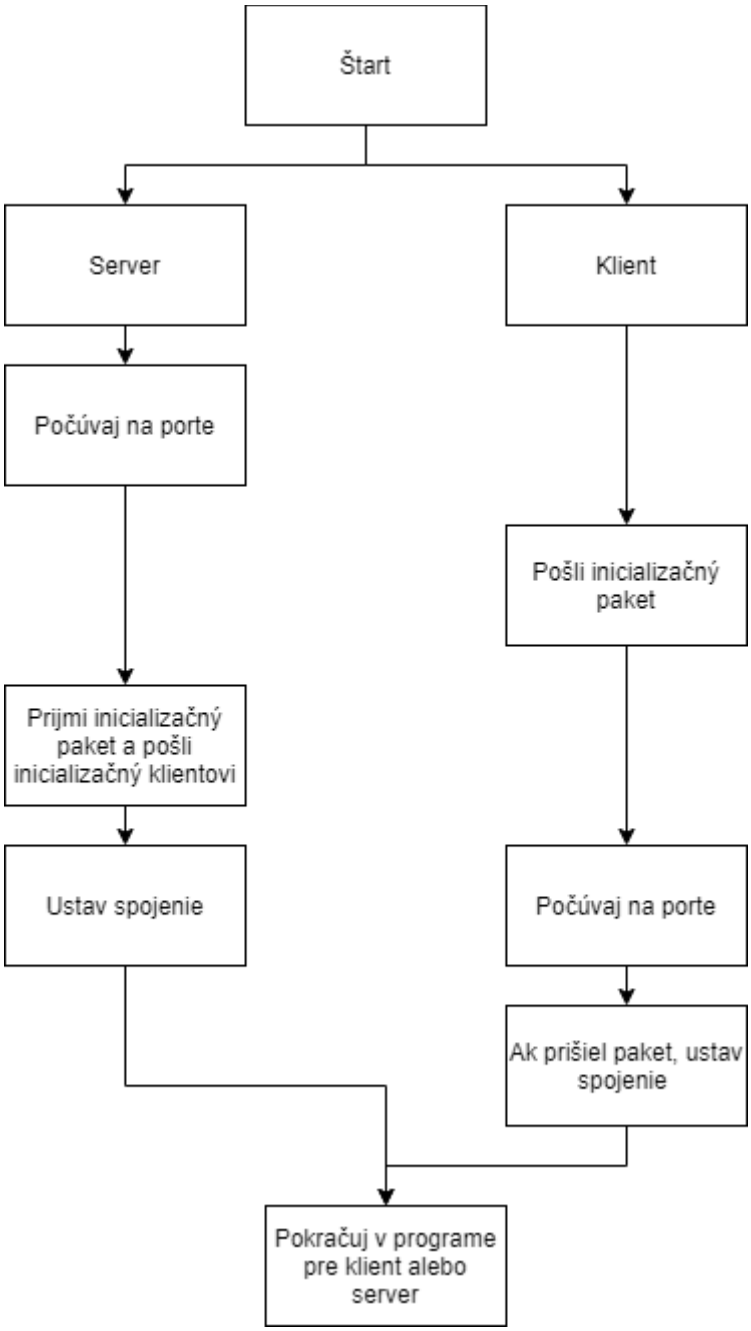


V poli ***Packet number*** bude poradie paketu. 3B preto, lebo ako som už spomenul vyššie, môžem mať 2 milióny paketov. V poli ***CRC*** – veľkosť 4B – sa bude nachádzať zvyšok po CRC metóde. To ako funguje, opíšem neskôr vo vlastnej sekcií. V poli ***Data*** budú už samotné dáta. Maximálna veľkosť je 1465B. Pretože nechceme aby sa fragmentovalo na linkovej vrstve, treba veľkosť nastaviť na 1526B – 26B – 20B – 8B a – 7B kvôli mojej hlavičke.

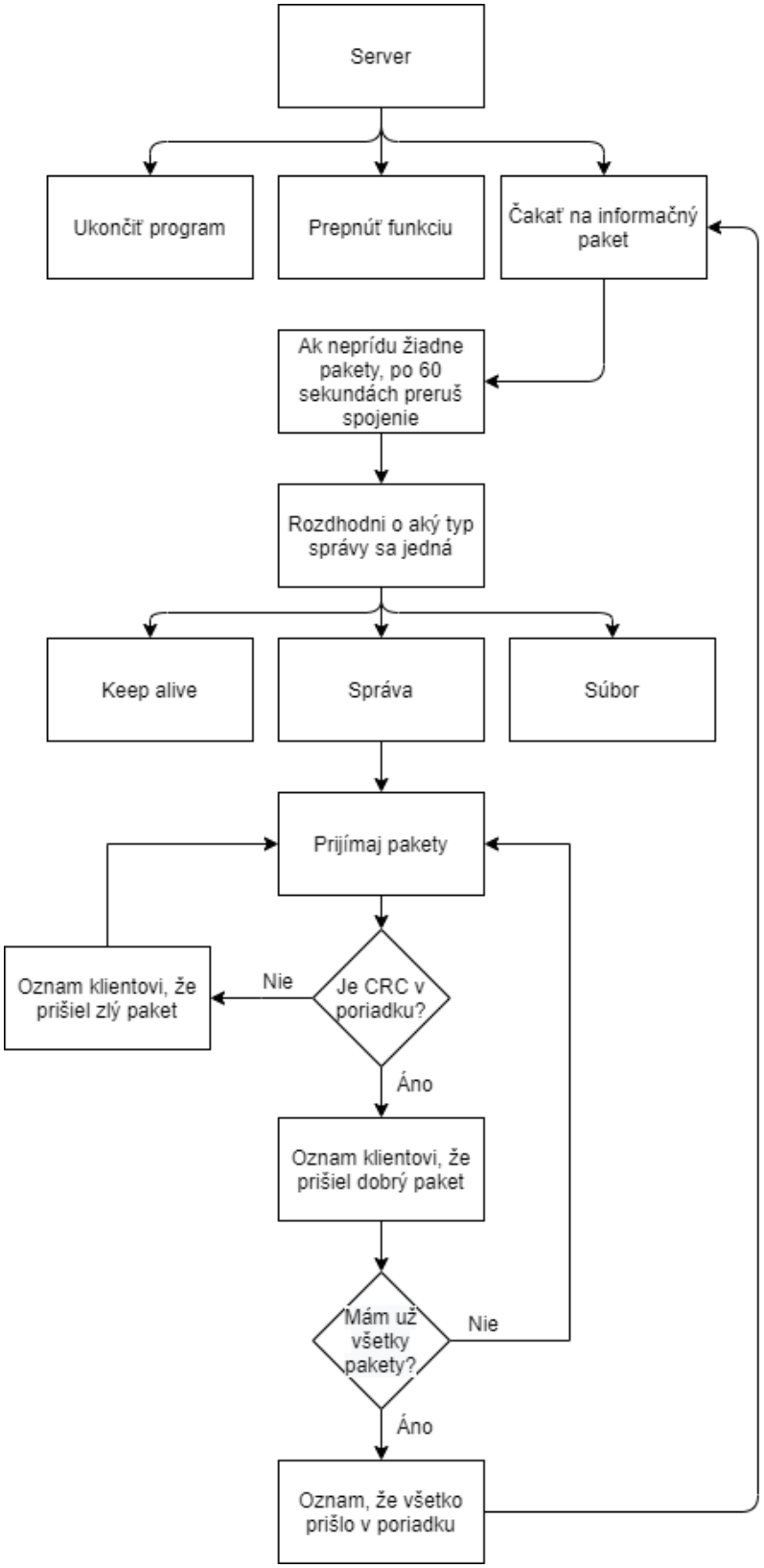
Matej Delinčák, ID: 102895

Message sequence diagram

V nasledujúcich diagramoch ukážem ako bude fungovať môj štart programu a samotné funkcie pre klient a server. Tieto diagramy nemusia byť vo finálnom odovzdaní rovnaké, môžu sa jemne líšiť.

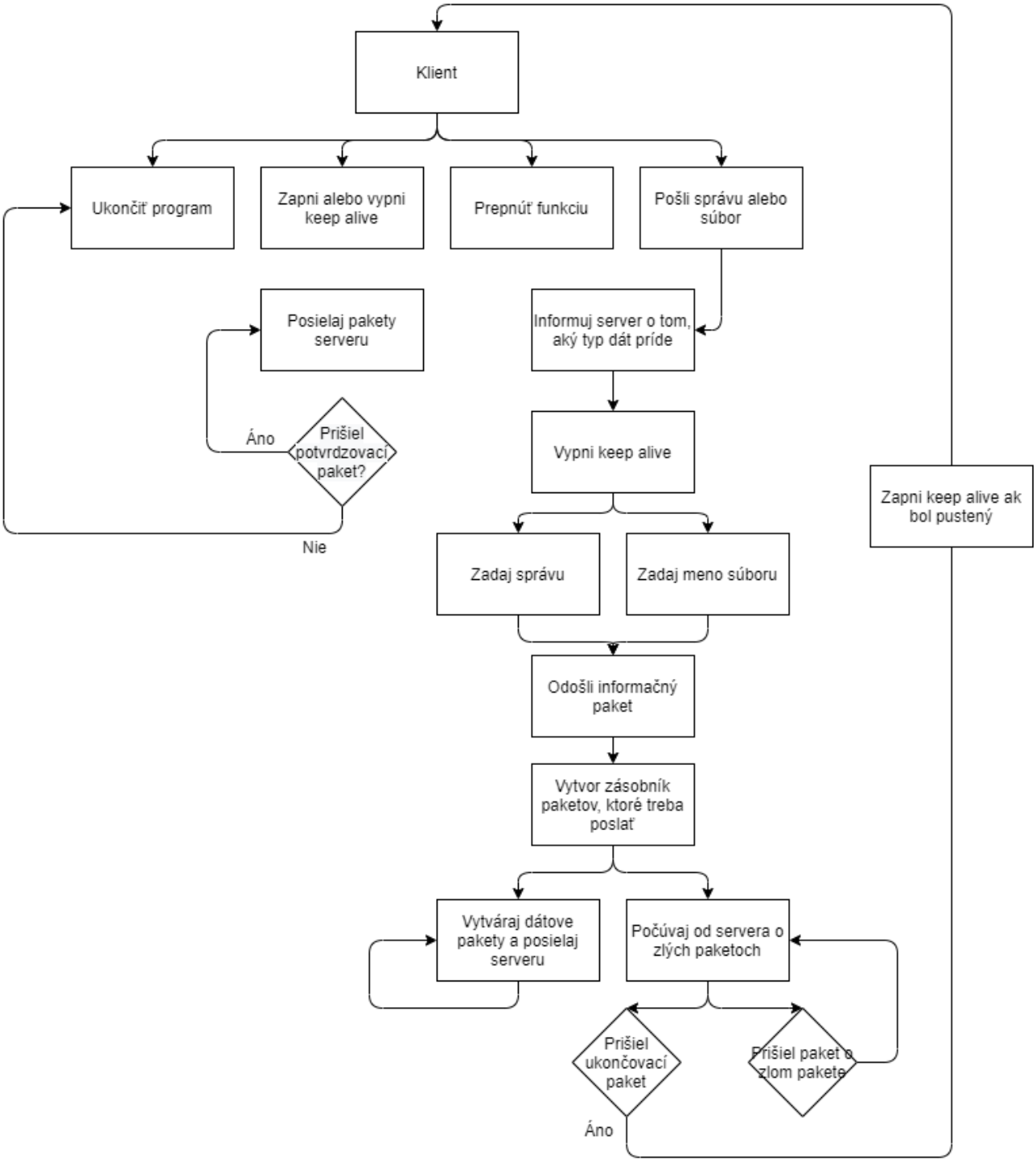


Matej Delinčák, ID: 102895



Matej Delinčák, ID: 102895

Klient pri výbere si bude môcť vybrať aj rozličnú veľkosť fragmentu.



Matej Delinčák, ID: 102895

ARQ metóda

Pre môj návrh som si vybral Selective ARQ metódu. Celá táto metóda spočíva v tom, že klient bude posielať zaradom dátové pakety zo zásobníka. Druhý thread zároveň počúva na odozvy od serveru. Server počúva a prijíma pakety. Ak prijme paket, ktorý je v poriadku, odošle klientovi správu o dobrom prijatí. Ak je paket zlý, pošle klientovi správu s číslom zlého paketu.

Klient teda tieto správy prijíma a rozhoduje sa na ich základe. Ak prišla pozitívne ACK, tak si zapíše, že tento paket prišiel v poriadku. Ak príde negatívne ACK, tak tento paket znova zaradí do zásobníka na odoslanie. Ak už všetko poslal a nemá všetky správy potvrdené, pošle pakety, ktoré asi ešte chýbajú serveru.

Na koniec, keď server má všetky pakety, pošle globálne ACK klientovi a ten prestane posielať ďalšie dáta.

Checksum algoritmus

Ako algoritmus pre kontrolu, či dáta neprišli porušené, použijem metódu CRC. A to konkrétne funkciu *crc32()* z knižnice *zlib*. Túto metódu som skúšal naprogramovať sám, len bola veľmi veľmi pomalá, tak som sa rozhodol pre funkciu z knižnice. A teda aby som ju mohol použiť, opíšem ako funguje.

vstup	0	1	0	0	0	0	0	1																																
obrateny vstup	1	0	0	0	0	0	1	0																																
vstup + 32 nul	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0xFFFFFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	
XOR	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	
polynom		1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1							
	0	0	1	1	1	1	0	0	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	
			1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1						
	0	0	0	1	1	1	0	0	0	1	0	1	0	1	1	1	1	0	0	1	1	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	
				1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1					
	0	0	0	0	1	1	0	0	0	0	0	1	1	0	1	1	1	0	0	0	1	0	1	0	1	1	1	1	1	1	1	1	0	1	0	1	0	0	0	0	
					1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1				
	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	1	1	0	1	0	0	0	
						1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	1			
	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	0	0	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0	1	1	0	1	0	0	
0xFFFFFFFF									1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
XOR	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	1	1	1	1	0	0	1	1	0	0	1	1	0	1	1	1	1	0	0	1	0	1	1	
obrateny vysledok									1	1	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	0	0	1	1	1	1	0	1	0	0	0	1	1	0	1	=CRC

Na vstupe mam pre príklad vstup „A“, ten prevediem na číslo v binárnej sústave a zrkadlovo ho obrátim. Pridám k dátam na koniec 32 núl a na takýto vstup použijem XOR s hodnotou 0xFFFFFFFF. Teraz prichádza hlavný cyklus crc, kde XORujem polynóm s medzivýsledkom. Keď prídem na koniec výsledok zase zoXORujem s 0xFFFFFFFF a zrkadlovo obrátim. Výsledok je hodnota CRC.

Funkcia zlib.crc32() používa polynóm $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x^{1} + 1$. Čo je v binárnej sústave 1 0000 0100 1100 0001 0001 1101 1011 0111.

Keep alive metóda

Ak klient pustí keep alive, tak každých 30 sekúnd sa pošle správa serveru. Ten keď obrdží túto správu, pošle ACK klientovi. Týmto sa bude udržiavať spojenie. Ak ale nepríde ACK, tak to klient vyhodnotí ako uzavretie spojenia. Na druhej strane server uzaviera spojenie, ak do 60 sekúnd nepríde nič.

Matej Delinčák, ID: 102895

Zmeny oproti návrhu

Vlastná hlavička

V mojich hlavičkách oproti návrhu došlo len k jednej zmene, a to, že v informatívnom pakete v poli ***Type*** pribudla ďalšia hodnota:

Strana klient:

„7“ – oznamuje serveru, že klient skončil a uzaviera spojenie.

Message sequence diagram

V hlavnom cykle programu, kde si užívateľ vyberá, či bude klient alebo server, som pridal možnosť program spustiť znova. Bez toho, aby bolo treba pustiť celý program znova.

Čo sa týka zmien pre klienta:

Keep alive: pre keep alive som dorobil časovač. Klient pošle keep alive serveru a server má dve sekundy na to, aby odpovedal. Ak neodpovie, dám užívateľovi informáciu, o tom, že server už asi nepočúva. Užívateľ môže potom ďalej pracovať, ale nič mu nepôjde. Čiže mu ostáva už len ukončiť spojenie.

Posielanie súborov/správ: predtým, než začnem posielať nejaké dáta, pošlem po novom jeden keep alive na zistenie, či druhá strana ešte počúva. Ak mi príde do dvoch sekúnd odpoveď, pokračujem v posielaní dát. Ak nie, zase oznámim užívateľovi, že niečo nie je v poriadku.

V pôvodnom návrhu som nerátal zo strácaním paketov. To som vyriešil takto. Najskôr naplním zásobník paketmi, ktoré treba poslať. Tieto všetky pošlem a keď mám prázdny zásobník, vyskakujem z cyklu. Za cyklom pol sekundy počkám a potom zistím, či som dostal na všetky pakety ACK. Ak nie, tak zásobník naplním číslami paketov, ktoré nemám potvrdené. Počas tohto celého mám pustený druhý thread, ktorý počúva ACK a NACK od servera, prípadne paket o úspešnom poslaní celých dát. Tento thread napĺňa už spomínaný zásobník. A maže čísla paketov, ktoré už netreba poslať, pretože dostal ACK. Tú pol sekundu, čo som spomínal vyššie, som tam dal na to, aby sa server stihol spamätať a aby som zbytočne neposlal paket, ktorý možno prišiel, ale nebol ešte potvrdený zo strany servera.

Vytváranie chybných paketov: v návrhu mi chýbalo opísané, ako budem vyrábať chybné pakety. A teda, klient si zadá, koľko chybných paketov chce a ja vyberiem n prvých paketov a tie pokazím. Chybu simulujem tak, že CRC vydelím na polovicu.

Zmeny na strane servera:

Strana servera ostala viac menej rovnaká ako som uviedol v návrhu a neudiali sa tu žiadne razantné zmeny. Avšak doplnil som funkciu, kde na začiatku programu si užívateľ vyberá, kam chce ukladať prijaté súbory. A po každom prijatí správy alebo súboru, sa opýta užívateľa, či chce skončiť program, pokračovať v počúvaní alebo prepnúť rolu v komunikácii.

Zároveň som zrobil aby server poslal ACK na každú správu čo dostane. Teda aj pre informačné pakety.

Matej Delinčák, ID: 102895

Implementačné prostredie

Ako implementačné prostredie som zvolil Python. Je to preto, lebo sa ľahko v ňom pracuje a má dostupných veľa knižníc. Zoznam knižníc, ktoré som použil a prečo:

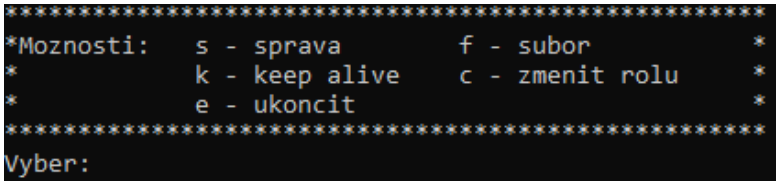
- **os** – pre prácu zo súbormi
- **socket** – pre prácu so socketmi a na prepojenie užívateľov, teda celý program stojí na tejto knižnici
- **struct** – pre prevádzanie dát na dátovy typ bytes
- **time** – z tejto knižnice som využil funkciu sleep() na uspatie napríklad threadu pre keep alive
- **threading** – vytváranie threadov, či už pri keep alive, alebo pri ARQ metode
- **zlib** – využil som funkciu crc32() na vyrátanie CRC pre dátové pakety

Kód som rozdelil do 5 skupín:

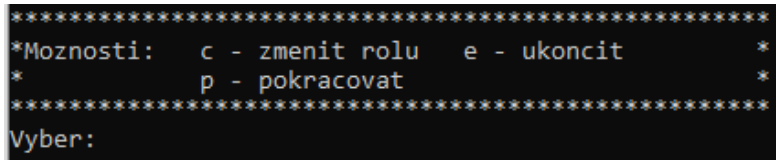
- Skupina *PROTOKOL* obsahuje funkcie na zakódovanie a dekódovanie informácií na dáta a naopak.
- Skupina *KEEP* ALIVE obsahuje jednu funkciu, čo je samotný keep alive a globálnu premennú na vypnutie keep alive-u.
- Skupina *CLIENT* obsahuje funkcie pre výpis možností pre užívateľa, posielanie dát či počúvanie odpovedí od servera. Ako aj počiatočnú funkciu na nadviazanie spojenia a hlavnú funkciu *main_client(),* v ktorej prebieha hlavný cyklus programu.
- Skupina *SERVER* obsahuje taktiež funkciu na výpis funkcionality pre stranu serveru, funkciu na začatie komunikácie a hlavný cyklus serveru sa nachádza vo funkcií *main_server().*
- A na koniec *HLAVICKY* kde sa nachádzajú pomocné funkcie na popis, v akej časti programu sa užívateľ nachádza.

Používateľské prostredie

Po spustení programu si užívateľ vyberá, či chce byť server alebo klient. Po úspešnom pripojení sa zobrazí správa, že sa tak vykonalo a pokračuje sa ďalej. Pre klienta sa zobrazí tabuľka možností, čo program dokáže:



Užívateľ teda môže poslať súbor alebo správu, kde vyberie veľkosť fragmentu a počet chybných paketov, zapnúť alebo vypnúť keep alive, zmeniť rolu alebo ukončiť komunikáciu.



A server má možnosti zmeniť rolu, ukončiť komunikáciu, alebo ak server má správne fungovať, treba stlačiť po každej prijatej správe alebo súbore „p“. Ak prichádzajú keep alive-y, tak po týchto sa server nepýta, či treba pokračovať alebo nie.

Ako je vidno tak obe strany majú možnosť si vymeniť rolu. Komunikácie tohto typu nie je nijako zabezpečená a každá strana sa musí manuálne prepnúť, aby program fungoval ďalej. Keď sa strana prepne, tak sa konzola vyčistí.

Záver

Na rozdiel od prvého zadania, ma toto veľmi bavilo. Dokázal som sa s ním vyhrať a poskúšal som rôzne ARQ metódy, aj keď som mal v návrhu len selective ARQ. Môj program som skúšal spustiť aj na dvoch počítačoch a všetko fungovalo tak, ako keď používam localhost. V zadaní je napísané, že máme vedieť poslať aj 2MB súbor, no keďže v hlavičke na počet paketov mam použité tri bajty, dokážem poslať aj väčší súbor. A to približne 16MB pri najväčšej veľkosti fragmentu. Taktiež vo wiresharku dokážem zachytiť moju komunikáciu a je presne taká, akú ju posielam. Nič sa tam nenachádza navyše. Keďže v mojom návrhu chýbali zdroje, tak ich tu ešte dodatočne uvediem. Ide hlavne o dokumentácie ku knižniciam a ku fungovaniu môjho crc.

Zdroje

https://docs.python.org/3/library/zlib.html

https://docs.python.org/3/library/socket.html

https://www.lammertbies.nl/comm/info/crc-calculation

https://www.anintegratedworld.com/how-to-calculate-crc32-by-hand/