Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológii

8. December 2021

Počítačové a komunikačné siete

Komunikátor s využitím

UDP protokolu

Roland Vdovják

id: 110912

Obsah

[Zadanie úlohy 2](#_Toc89890822)

[Návrh riešenia 4](#_Toc89890823)

[Štruktúra hlavičky 4](#_Toc89890824)

[Typ 4](#_Toc89890825)

[Veľkosť 4](#_Toc89890826)

[Poradie 4](#_Toc89890827)

[CRC 4](#_Toc89890828)

[ARQ 5](#_Toc89890829)

[Udržiavanie spojenia 5](#_Toc89890830)

[Diagram spracovávania 6](#_Toc89890831)

[Prijímajúca strana 6](#_Toc89890832)

[Odosielajúca strana 7](#_Toc89890833)

[Riešenie 8](#_Toc89890834)

[Zmena oproti návrhu 8](#_Toc89890835)

[CRC 8](#_Toc89890836)

[Princíp 9](#_Toc89890837)

[Fungovanie programu 9](#_Toc89890838)

[Spojenie Klient-Server 9](#_Toc89890839)

[Klient 10](#_Toc89890840)

[Server 11](#_Toc89890841)

[Keep Alive 11](#_Toc89890842)

[Možnosti ukončenia spojenia 11](#_Toc89890843)

[Posielanie 11](#_Toc89890844)

[Chyba v dátovej časti 13](#_Toc89890845)

[Prijímanie 13](#_Toc89890846)

[Užívateľské rozhranie 14](#_Toc89890847)

[Diagramy 17](#_Toc89890848)

[Wireshark 20](#_Toc89890849)

# 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 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.

Program musí mať nasledovné vlastnosti (minimálne):

1. Program musí byť implementovaný v jazykoch C/C++ alebo Python s využitím knižníc na prácu s UDP socket, skompilovateľný a spustiteľný v učebniach. Odporúčame použiť python modul socket, C/C++ knižnice sys/socket.h pre linux/BSD a winsock2.h pre Windows. Iné knižnice a funkcie na prácu so socketmi musia byť schválené cvičiacim. V programe môžu byť použité aj knižnice na prácu s IP adresami a portami:

arpa/inet.h

netinet/in.h

2. Program musí pracovať s dátami optimálne (napr. neukladať IP adresy do 4x int).

3. Pri posielaní súboru musí používateľovi umožniť určiť cieľovú IP a port.

4. Používateľ musí mať možnosť zvoliť si max. veľkosť fragmentu.

5. Obe komunikujúce strany musia byť schopné zobrazovať:

a. názov a absolútnu cestu k súboru na danom uzle,

b. veľkosť a počet fragmentov.

6. Možnosť simulovať chybu prenosu odoslaním minimálne 1 chybného fragmentu pri

prenose súboru (do dátovej časti fragmentu je cielene vnesená chyba, to znamená, že

prijímajúca strana deteguje chybu pri prenose).

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

8. Možnosť odoslať 2MB súbor a v tom prípade ich uložiť na prijímacej strane ako rovnaký súbor, pričom používateľ zadáva iba cestu k adresáru kde má byť uložený.

Odovzdáva sa:

1. Návrh riešenia

2. Predvedenie riešenia v súlade s prezentovaným návrhom

Program musí byť organizovaný tak, aby oba komunikujúce uzly mohli prepínať medzi funkciou vysielača a prijímača bez reštartu programu - program nemusí (ale môže) byť vysielač a prijímač súčasne. Pri predvedení riešenia je podmienkou hodnotenia schopnosť doimplementovať jednoduchú funkcionalitu na cvičení.

# Návrh riešenia

## Štruktúra hlavičky

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Typ | Veľkosť | Poradie | CRC | Dáta |
| 1B | 2B | 2B | 4B | ... |

### Typ

Hodnota TYP určuje buď jeden alebo viac flagov, ktoré daný opisujú daný rámec. Flagy môžu byť:

|  |  |
| --- | --- |
| Bajt [bin0] | Typ flagu (správy) |
| 0000 0001 | Nadviazanie spojenia |
| 0000 0010 | Prijatie fragmentu |
| 0000 0100 | Odmietnutie fragmentu |
| 0000 1000 | Keep alive |
| 0001 0000 | Ukončenie spojenia |
| 0010 0000 | Posielanie dát |
| 0100 0000 | Potvrdenie spojenia |

### Veľkosť

Veľkosť fragmentu, ktorý sa prenáša. Hovorí o reálnej veľkosti, ktorá sa prenáša. Keď má naša hlavička 9B, IP hlavička 20B, UDP hlavička 8B, tak veľkosť bude maximálne

Veľkosť je stanovená práve takto, aby nebolo nutné fragmentovať na druhej vrstve modelu OSI.

### Poradie

Poradie fragmentu.

### CRC

Implementovali funkciu *crc32()* z knižnice *zlib*. Veľkosť návratovej hodnoty funkcie *crc32()* sú štyri bajty, preto v hlavičke zaberá práve 4B.

## ARQ

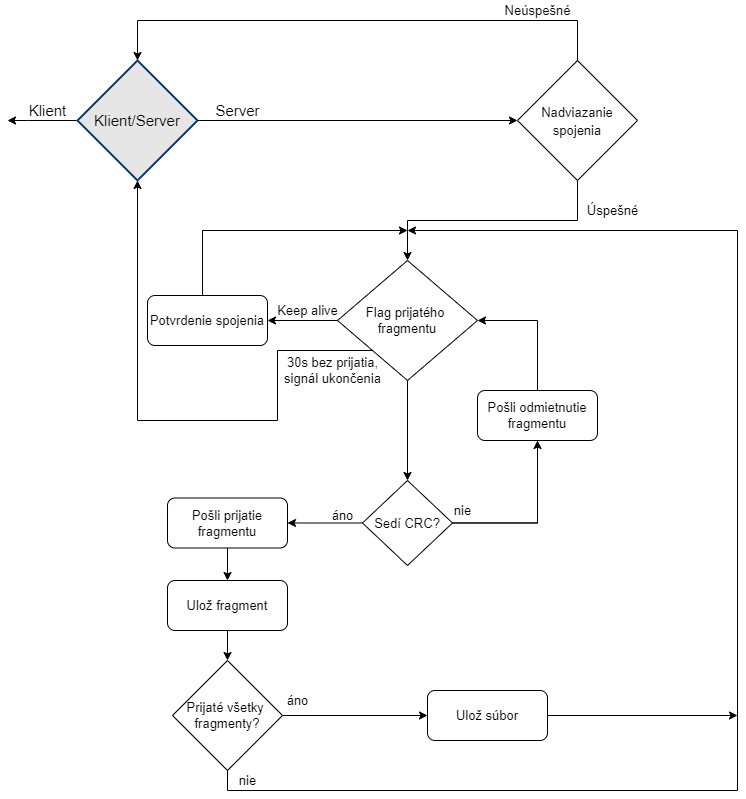
Stop and wait – táto metóda znamená, že po každom jednom fragmente čaká klient na odpoveď servera, či bol fragment prijatý alebo nie. Ak bol prijatý, tak sa pokračuje ďalším fragmentom až kým nepošleme celý súbor, ktorý odosielame. Ak nebol fragment prijatý tak klient posiela znovu rovnaký fragment, až kým nie je prijatý.

## Udržiavanie spojenia

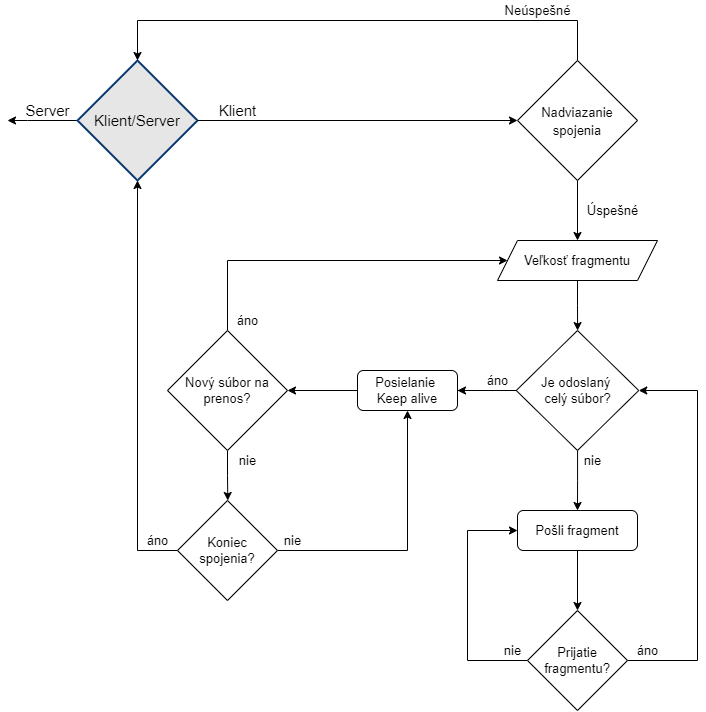
Udržiavame spojenie pomocou odosielania s typom flagu „Keep alive“. Tento fragment odosielame každých 10 sekúnd. Ak po 30 sekundách neprišiel Keep alive flag, tak sa ukončuje spojenie. Iná možnosť je ukončiť spojenie klientom po úspešnom odoslaní súboru bez potreby ďalšieho odosielania. Aby klient vedel, že je server stále dostupný, tak bude server odpovedať na správu Keep alive potvrdením spojenia, bez odpovede do niekoľkých sekúnd program ukončuje spojenie.

## Diagram spracovávania

### Prijímajúca strana



### Odosielajúca strana



# Riešenie

## Zmena oproti návrhu

Zmena v hlavičke, v type (vo flagu)

|  |  |
| --- | --- |
| Bajt [bin0] | Typ flagu (správy) |
| 0000 0001 | Nadviazanie spojenia |
| 0000 0010 | Prijatie fragmentu |
| 0000 0100 | Odmietnutie fragmentu |
| 0000 1000 | Keep alive |
| 0001 0000 | Ukončenie spojenia |
| 0010 0000 | Potvrdenie spojenia |
| 0100 0000 | Posielanie dát |
| 0100 0001 | Názov súboru |
| 1000 0000 | Posielanie textu |

Okrem tejto zmeny sme nemenili návrh.

## CRC

Implementácia *crc* je pomocou knižnice *zlib*. Konkrétne ide o *crc32*. Polynóm určený pre túto funkciu je:

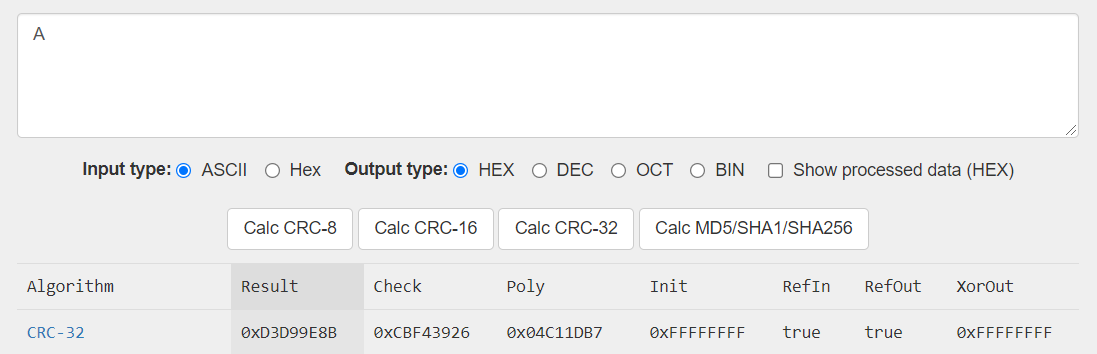
HEX: BIN:

Príklad riešenia pre input „A“:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| input | 0100 | 0001 |  |  |  |  |  |  |  |  |
| reversed | 1000 | 0010 |  |  |  |  |  |  |  |  |
|  | 1000 | 0010 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 0xFFFFFFFF | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 |  |  |
| XOR | 0111 | 1101 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 0000 | 0000 |
| polynom | 100 0001 0011 0000 0100 0111 0110 1101 11 | | | | | | | | | |
| XOR | 0011 | 1100 | 1100 | 1111 | 1011 | 1000 | 1001 | 0010 | 1100 | 0000 |
| polynom | 10 0000 1001 1000 0010 0011 1011 0110 111 | | | | | | | | | |
| XOR | 0001 | 1100 | 0101 | 0111 | 1001 | 1011 | 0010 | 0100 | 0010 | 0000 |
| polynom | 1 0000 0100 1100 0001 0001 1101 1011 0111 | | | | | | | | | |
| XOR | 0000 | 1100 | 0001 | 1011 | 1000 | 1010 | 1111 | 1111 | 0101 | 0000 |
| polynom | 1000 0010 0110 0000 1000 1110 1101 1011 1 | | | | | | | | | |
| XOR |  | 0100 | 0011 | 1101 | 1000 | 0010 | 0001 | 0010 | 1110 | 1000 |
| polynom | 100 0001 0011 0000 0100 0111 0110 1101 11 | | | | | | | | | |
| XOR |  | 0000 | 0010 | 1110 | 1000 | 0110 | 0110 | 0100 | 0011 | 0100 |
| 0xFFFFFFFF |  |  | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 |
| XOR |  |  | 1101 | 0001 | 0111 | 1001 | 1001 | 1011 | 1100 | 1011 |
| reversed |  |  | 1101 | 0011 | 1101 | 1001 | 1001 | 1110 | 1000 | 1011 |

Tabuľka : Vypočítanie CRC32

Výsledok prevedený do hexadecimálneho formátu je 0xd3d99e8b. Ako kontrolu správnosti sme použili internetovú stránku (<https://crccalc.com/> ) na výpočet crc32.



### Princíp

Vstup sa premení na binárne číslo, obráti sa a pridá sa k nemu 32 núl (oranžový riadok tabuľky). Toto číslo je začiatkom XOR operácií. Prvý a posledný XOR, teda XOR z upraveného vstupu a z 32 bitového zvyšku je vypočítaný s hodnotou 0xFFFFFFFF (modré riadky). Hodnoty medzi prvým a posledným XOR-om sú počítané s polynómom. Výsledkom je 32 bitové číslo.

## Fungovanie programu

Všetky časti programu sú dostatočne okomentované v kóde, pre prehľadnosť dokumentácie v nej spomenieme len dôležité časti kódu a logiku programu.

### Spojenie Klient-Server

Používateľ by si mal najskôr vybrať rolu server, zadať port na ktorom bude počúvať (ak sa tak nestane, pripojenie nebude korektné a na strane klienta bude potrebné zadať znovu IP adresu a port). Potom na druhom zariadení používateľ vyberie rolu klient, zadá IP adresu servera a port.

socket()

**Server**

**Klient**

socket()

bind(“ ”, port)

sendto()

recvfrom()

recvfrom()

sendto()

Začatie spojenia

Začatie spojenia

Ak neprišla

odpoveď

skúsi znovu

Ak neprišla

odpoveď

skúsi znovu

Úspešné spojenie

Úspešné spojenie

Obrázok 1: Počietočné spojenie Klient-Server

Na signál dobrého spojenia musí klient poslať prázdnu hlavičku len s flagom začiatku spojenia a server mu vráti rovnakú prázdnu hlavičku len s flagom. Server ďalej počúva prichádzajúce rámce a správa sa podľa [diagramu](#_Diagramy). Každé rozhodnutie v diagrame reprezentuje číslo flagu v hlavičke. Keďže nie je pri posielaní textu alebo dát špecifikovaný flag posledného fragmentu, ktorý klient odosiela, server zisťuje, či je odoslané všetko vďaka kombinácií po sebe idúcich fragmentov s flagom Posielam text alebo Posielam dáta a Keep Alive.

### Klient

Klient sa pripája na server pomocou zadania IP serveru a portu, požívateľ zadá po výbere role klient. Po úspešnom spojení sa začína posielať hlavička s flagom Keep alive. Po spojení má na výber z troch možností:

* Poslať súbor [1]
  + Program si vypýta cestu ku súboru a veľkosť jedného fragmentu, pošle počiatočný rámec s meno súboru. Následne posiela fragmenty súboru. Po odoslaní vypíše informáciu o poslaní.
* Poslať text [2]
  + Program očakáva textový input. Implementovaný je multiline string. Na ukončenie inputu textu je potrebné zadať na riadok len znak „$“. Potom si program vypýta veľkosť fragmentu a posiela text.
* Vymeniť rolu/Ukončiť [3]
  + Zastaví vlákno s KeepAlive, pošle rámec s flagom na ukončenie spojenia, ukončí spojenie a presmeruje používateľa na výber Server/Klient/Ukončiť.

### Server

Po vybratí role server je potrebné zadať port na ktorom bude server počúvať akonáhle sa na tento port pripojí klient. Po spojení s klientom čaká na input ktorý rozhoduje o:

* Pokračovať
  + Server je pripravený na počúvanie
* Vymeniť rolu/Ukončiť [0]
  + Odošle rámec s s flagom na ukončenie spojenia, zatvorí socket a presmeruje používateľa na výber Server/Klient/Ukončiť.
* Pokračovať ako server [1]
  + Čaká na pripojenie klienta, port zostáva rovnaký

Ak server pokračuje, je pripravený prijímať rámce. Po každom prijatom rámci vyvodí nejakú akciu. Po prenesení súboru alebo textu má používateľ na výber možnosti spomínané vyššie.

### Keep Alive

Implementácia posielania rámcov s keep alive je v samostatnom vlákne v klientovi. Na strane servera sú prijímané rámce s keep alive rovnako ako všetky ostatné. Ak na server príde keep alive, server odpovedá „Connection confirmed“. Ak klient nedostane odpoveď do desiatich sekúnd, pošle znovu. Toto opakuje trikrát. Ak po tridsiatich sekundách neprišla odpoveď, klient vyhodnotí, že server neodpovedá a naviguje klienta na ukončenie.

### Možnosti ukončenia spojenia

Program sa môže ukončiť viacerými spôsobmi, nie všetky sú korektné, ale sú ošetrené.

* Ukončenie zo strany klienta- odoslaný „Connection end“ flagu
* Ukončenie zo strany servera- odoslaný „Connnection end“ flagu
* Odpojenie klienta- bez odoslania „Connection end“ flagu
  + Program s klientom nečakane skončí, odpojí sa zo siete, prestane posielať

Keep Alive správy.

* Odpojenie servera- bez odoslaný „Connection end“ flagu
  + Program so serverom nečakane skončí, odpojí sa zo siete.

Ukončovací flag má hodnotu 16 a je pomenovaný ako „Connection end“, ak tento pragment prijme server alebo klient, používateľ dostane o tom informáciu a je navigovaný ako pokračovať.

### Posielanie

Na posielanie je určená jedna funkcia, ktorá ako parameter berie kľúč, ktorý hovorí o tom, či sa posiela text alebo súbor. Hlavným rozdielom medzi posielaním textu a súboru je tvorba počiatočných premenných. Pri posielaní súboru je potrebné zistiť zo vstupu o aký súbor sa jedná, zistiť jeho veľkosť, dáta uložiť do premennej vo forme bajtov. Ďalej názov bez cesty, ktorý sa rovno pošle na server a stanoviť flag fragmentu, ktorý sa bude odosielať pri posielaní dát.

Ukážka kódu:

**# Posiela sa subor  
if f\_type == '1':  
 f\_path = input("Zadaj cestu k suboru.\n")  
 f = open(f\_path, 'rb')  
 f\_name = (os.path.basename(f\_path)).encode('utf-8')  
  
 # odoslanie mena suboru  
 h\_type = 65  
 h\_crc = crc(f\_name)  
 h\_frag\_num = 0  
 h\_size = len(f\_name)  
 frag\_full = h\_type.to\_bytes(1, byteorder='big') + h\_size.to\_bytes(2, byteorder='big') + h\_frag\_num.to\_bytes(2, byteorder='big') + h\_crc.to\_bytes(4, byteorder='big') + f\_name  
 print("Posielam meno suboru")  
 client\_s.sendto(frag\_full, server\_address) # Odoslanie  
  
 size = os.path.getsize(f\_path)  
 data = f.read()  
 h\_type = 64  
  
# Posiela sa textova sprava  
if f\_type == '2':  
 print("Zadaj text spravy.\n")  
 text\_msg = load\_text\_msg()  
 data = text\_msg.encode('utf-8')  
 size = len(data)  
 h\_type = 128**

Premenné s predponou „*h­\_*­“ hovoria o hlavičke. *F\_type* je typ súboru- teda text alebo súbor.

Po tejto počiatočnej časti program zisťuje veľkosť fragmentu od používateľa, táto veľkosť je obmedzená. Môže vyť v intervale *<1, 1463>* a zároveň počet fragmentov musí byť menší ako 65535. Dôvodom je veľkosť hodnoty Poradie v hlavičke, táto veľkosť je ohraničená na 2B. Ak by po zadaní veľkosti fragmentu bol počet fragmentov väčší ako 65534, tak musí používateľ zadať veľkosť znovu, používateľ je inštruovaný minimálnou potrebnou veľkosťou.

Po tomto vchádza program do cyklu odosielania, každým opakovaním sa vytvorí hlavička a odreže potrebná časť dát, všetky údaje sa spoja a pošlú. Po poslaní očakávajú fragment hovoriaci o tom, či fragment prešiel bez problémov alebo bol odmietnutý. Podľa toho pokračuje ďalej alebo znovu odosiela pôvodne odmietnutý fragment.

Po odoslaní celého súboru alebo textu vypíše, že súbor alebo text bol odoslaný a jeho veľkosť.

### Chyba v dátovej časti

Z dôvodu nutnosti posielať chybu v dátovej Časti sme sa rozhodli posielať prvý resp. nultý bajt rámca ako samé nuly. Pôvodný bajt si odložíme do premennej *original\_byte*. Výber toho, ktoré rámce budú týmto ovplyvnené nastavujeme v globálnej premennej. Posielame chybných pár rámcov v prvej desiatke odoslaných. Potenciálne vylepšenie sú jednoduché, len meníme obsah globálnej premennej- poľa.

Po odoslaní chyby príde na príjem klienta flag rámca „Fragment redjected“. Po zaznamenaní tohto flagu vrátime hodnotu bajtu z premennej *original\_byte* do nultého bajtu dátovej časti.

Nultý index je stanovený z dôvodu posielania minimálne jedného bajtu pri posielaní súboru alebo textu.

Ukážka kódu:

**# Posielanie chyb  
if sended and h\_frag\_num in wrong\_fragments:  
 original\_byte = frag\_data[0]  
 frag\_data[0] = 0**

Premenná *frag\_data* obsahuje fragment dát daného rámca. *Sended* hovorí o tom, či bol predchádzajúci fragment odoslaný. *Wrong\_fragments* je globálna premenná typu poľa. A *h\_frag\_num* je momentálne číslo rámca ktorý sa posiela.

### Prijímanie

Server prijíma rámce a podľa hodnoty flagu sa rozhoduje, akú akciu vykoná. Samotný try exept blok pozostáva z viacerých if blokov.

Ukážka kódu:

**# Vypis flagu**

**if type\_int != 8:  
 ...  
  
# Ukoncenie spojenia  
if type\_int == 16:  
 ...**

**# KEEP ALIVE + odpoved  
if type\_int == 8:  
 ...  
   
# Meno suboru  
if type\_int == 65:  
 ...  
   
# Kontrola crc  
if type\_int == 64 or type\_int == 128:  
 ...  
   
# Odoslanie celeho suboru  
if type\_int == 8 and (last\_type\_int == 64 or last\_type\_int == 128):  
 ...**

### Užívateľské rozhranie

Používateľ pracuje s programom cez terminál. Zadáva na vstup rôzne čísla alebo údaje, podľa ktorých program rozhoduje, čo bude robiť.

#### Menu

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok : Hlavné menu

#### Role

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok : Výber role Klient

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok : Výber role Server

#### Posielanie súboru

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok : Posielanie súboru na strane Klienta

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok : Odoslanie všetkých rámcov na strane Klienta

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok : Odoslanie všetkých rámcov na strane Servera

#### Posielanie textu

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok : Posielanie textu ma strane Klienta

Obrázok, na ktorom je text

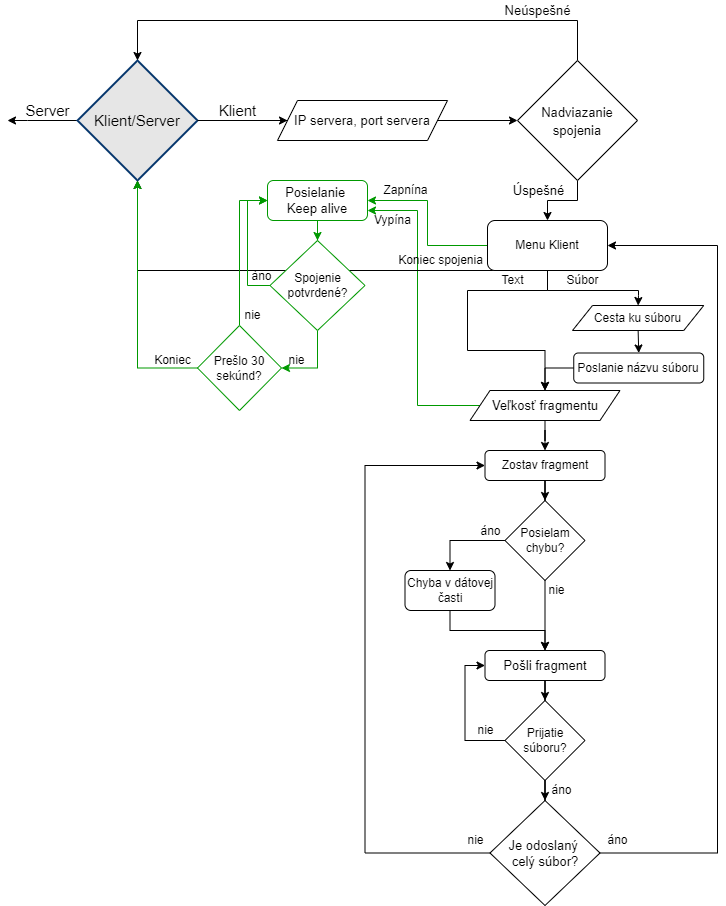
Automaticky generovaný popis

Obrázok : Posielanie textu na strane Servera

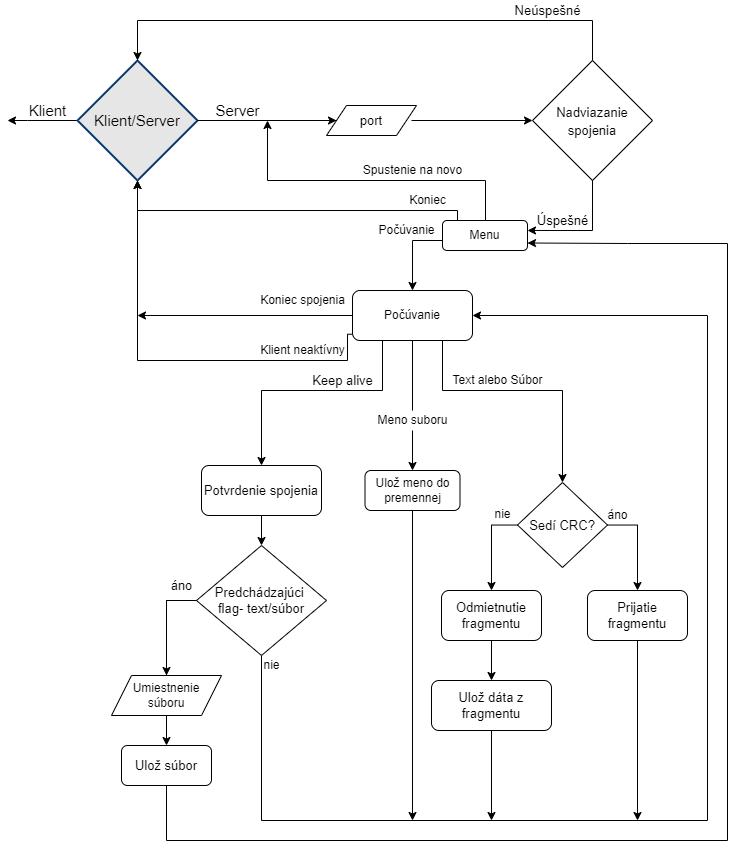
### Diagramy

Diagramy zostali oproti návrhu logicky rovnaké, no pridali sme niekoľko potrebných funkcií, vstupov, výstupov. Pri konzultácií sme dostali podnet na vylepšenie a spresnenie diagramov. Vylepšené diagramy:

Zelená časť diagramu reprezentuje samostatné vlákno.



Obrázok : Diagram Klient



Obrázok : Diagram Server

## Wireshark

Pri testovaní a debugovaní programu sme používali aj program Wireshark. Posielali sme rámce cez router na druhé zariadenie. Záznam z wiresharku sme vyfiltrovali aby nám ukazoval komunikáciu medzi IP adresami týchto zariadení. Záznam vyzerá takto. Ako legendu ku farbám používame rozšírenú tabuľku s flagmi.

Obrázok, na ktorom je stôl

Automaticky generovaný popis

Obrázok 12: Wireshark záznam- prenášaný text

|  |  |  |
| --- | --- | --- |
| Bajt [bin0] | Typ flagu (správy) | Farba |
| 0000 0001 | Nadviazanie spojenia |  |
| 0000 0010 | Prijatie fragmentu |  |
| 0000 0100 | Odmietnutie fragmentu |  |
| 0000 1000 | Keep alive |  |
| 0001 0000 | Ukončenie spojenia |  |
| 0010 0000 | Potvrdenie spojenia |  |
| 0100 0000 | Posielanie dát |  |
| 0100 0001 | Názov súboru |  |
| 1000 0000 | Posielanie textu |  |

Obrázok, na ktorom je stôl

Automaticky generovaný popis

Obrázok 13: Wireshark záznam- prenášaný súbor