SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta informatiky a informačných technológií  
Ilkovičova 2, 842 16 Bratislava 4

**Zadanie č. 2 – Komunikácia s využitím UDP protokolu**

POČÍTAČOVÉ A KOMUNIKAČNÉ SIETE

Róbert Junas

FIIT STU

Cvičenie: Štvrtok 14:00

24.11.2021

# Zadanie

**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 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*

1. Program musí pracovať s dátami optimálne (napr. neukladať IP adresy do 4x int).
2. Pri posielaní súboru musí používateľovi umožniť určiť cieľovú IP a port.
3. Používateľ musí mať možnosť zvoliť si max. veľkosť fragmentu.
4. Obe komunikujúce strany musia byť schopné zobrazovať:
   1. názov a absolútnu cestu k súboru na danom uzle,
   2. veľkosť a počet fragmentov.
5. 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).
6. Prijímajúca strana musí byť schopná oznámiť odosielateľovi správne aj nesprávne doručenie fragmentov. Pri nesprávnom doručení fragmentu vyžiada znovu poslať poškodené dáta.
7. 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.

**Hodnotenie**

Celé riešenie - max. 20 bodov (min. 7), z toho:

- max. 5 bodov za návrh riešenia;

- max. 1 bod za doplnenú funkčnosť (doimplementáciu) priamo na cvičení v požadovanom termíne podľa harmonogramu cvičení; V prípade, ak študent nesplní úlohu zadanú priamo na cvičeniach, nehodnotí sa výsledné riešenie;

- max. 14 bodov za výsledné riešenie.

Návrh a zdrojový kód implementácie študent odovzdáva v elektronickom tvare do AISu v určených termínoch.

# Návrh

## Hlavička

0 7 15 31

|  |  |  |
| --- | --- | --- |
| TYPE | FLAG | SIZE |
| SEQ\_NUM | | FRAG\_COUNT |
| DATA | | |
| CHECKSUM | |

Rámec môže mať niekoľko typov, ktoré určujú ako sa k nim bude program správať:

* Control – 0x00 – na kontrolné rámce, ako ukončenie, udržanie spojenia alebo na zistenie či je cieľ dosiahnuteľný,
* Text – 0x01 – na posielanie textovej správy,
* File – 0x02 – posielanie súboru.

Bajt, v ktorom je pole flag, určuje o akú správu sa jedná:

* SYN – 0x01
* FIN – 0x02
* ACK – 0x04
* RESEND – 0x08
* KEEP-ALIVE – 0x10
* SWAP – 0x20

FRAG\_COUNT a SEQ\_NUM majú veľkosť 2 bajty, čo pri maximálnej veľkosti dát 500 bajtov dokáže preniesť 65535\*500 = ~32,767 MB, čo je dostačujúce na zadanie.

## Kontrola chýb

Kontrola chýb, resp. checksum pridaný na koniec posielaného rámca, bude vykonávaná pomocou CRC16 s polynómom 0x1021 (4129), resp. 0x11021. Táto funkcionalita bude zložená z dvoch funkcií – jednej na overenie korektnosti prijatých dát a druhej pre vytvorenie kontrolnej hodnoty.

Výpočet bude prebiehať tak, že sa na koniec dát, ktoré chceme zahashovať sa pridajú 2 bajty, ktoré budú reprezentovať checksum. Následne bude v dátach hľadať prvý bit nastavený na 1 a od neho urobí XOR s polynómom a uloží sa do výsledku. Následne sa výsledku budú pridávať ďalšie bity z dát a ak výsledok bude od svojej prvého bitu nastaveného na 1 bude dlhý ako polynóm, tak sa znovu vykoná XOR operácia a zapíše sa do výsledku. Táto operácia sa bude opakovať pokiaľ neprejdeme všetky bity v dátach. Na koniec dát sa pridá vypočítané crc.

Kontrola prebehne rovnako s tým, že sa proces vykoná pre celý prijatý rámec aj s crc a ak po vykonaní výpočtu budú posledné bity (tie na ktorých je uložené crc) samé 0, tak rámec prišiel v poriadku. Inak nebol v poriadku a vyžiada, aby bolo okno poslané znovu.

## ARQ

ARQ metóda použijeme Go Back-N protokol, ktorý si udržiava posuvné okno N rámcov, ktoré musia byť odoslané. Toto okno znovu posiela N rámcov v prípade, že pre prvý rámec v okne neprišlo od prijímača potvrdenie ACK, resp. pri prípadoch, že správa nepríde na prijímač, bude poškodená alebo sa odpoveď ACK stratí. Znovu poslanie bude riešené časovačom na strane vysielača a to práve časovačom, ktorý bude nastavený na pár sekúnd. Ak vysielač dostane odpoveď, tak sa celé okno posunie, časovač sa zresetuje, odošle rámec, ktorý pribudol do okna a čaká na ACK pre ďalší rámec. Ak prídu rámce v zlom poradí, tak rámce budú ignorované, teda ich prijímač nezoberie – neodošle pre nich ACK správu.

Ďalej naša implementácia použije aj signalizačnú správu NACK, resp. RESEND + ACK, ktorá by ihneď odoslala celé okno znova, bez čakania či vlastne.

## Udržiavanie spojenia

Po prenesení súboru na prijímač sa spustí posielanie KEEP-ALIVE rámcov, ktoré sa budú posielať každých 10 sekúnd. Ak odpoveď na tieto správy nepríde do 3 pokusov, tak sa spojenie bude brať za ukončené. Rovnako, ak do 10 sekúnd od poslania posledného rámca KEEP ALIVE alebo posledného rámca prenosu dát nepríde ďalší KEEP ALIVE rámec tak sa spojenie preruší. Prípadne sám prijímač alebo vysielač vyšlú požiadavku na ukončenie spojenia alebo na poslanie ďalšieho súboru, v oboch prípadoch sa prestanú posielať KEEP-ALIVE správy.

## Fragmentácia

Môže sa stať, že chceme poslať dáta, ktoré sú väčšie ako je maximálne daná veľkosť jedného rámca. Pri takýchto prípadoch sa správa rozfragmentuje na viac častí, ktoré sa odosielajú po poradí prijímaču, resp. dajú sa do radu packetov, čakajúcich na odoslanie. Následne sa v prijímači poskladajú v poradí akom prišli, pretože nami zvolená ARQ metóda zaručuje, že rámce prídu v poradí akom majú.

Maximálna veľkosť fragmentu bude podobná ako pri TFTP, kde každý prenášaný blok dát má hodnotu najviac 512 bajtov a veľkosť hlavičky je 4 bajty. Pre náš protokol to bude fungovať podobne s tým, že máme väčšiu hlavičku (16 bajtov), teda maximálna veľkosť fragmentovaného bloku bude ~500 bajtov a minimálna bude 1 bajt (pre prenos 2MB súboru by minimum malo byť okolo 40 Bajtov). Rovnako sme sa dozvedeli, že najbezpečnejšia veľkosť pre posielanie nad IPv4 je z 576 bajtov a posielanie väčších rámcov je v poriadku, ak vieme zaručiť, že sme ho schopný preniesť (https://www.rfc-editor.org/rfc/pdfrfc/rfc791.txt.pdf). Preto viac ako 576 bajtov nebudeme ani podporovať.

## Výmena rolí

Pri výmene rolí pošle vysielač prijímaču požiadavku na výmenu pomocou flagu SWAP, ktorý po prijatí zmení svoj stav z počúvania na posielanie a odošle potvrdenie o výmene role, keď vysielač príjme odpoveď tak zmení svoj stav.

Pred samotnou výmenou musí vysielač poslať aspoň jednu správu na nadviazanie spojenia s prijímačom.

## Začiatok spojenia

Pri začiatku spojenia sa pridá SYN rámec do radu a nastaví sa veľkosť okna na 1 rámec, dokým nepríde potvrdzovací rámec od prijímača. Po tom ako príde potvrdzovací rámec, tak sa zväčší okno na viac rámcov. Zároveň sa prijímač nastaví aby neprijímal rámce od iných zariadení.

## Časti kódu

### Knižnice

* Socket
* Threading
* Time – potrebné pre časovač nad ARQ

### Prijímač

* Zadať cieľový port, IP adresu a maximálnu veľkosť fragmentu
* Poslať korektnú správu ako aj simulovať chybu
* vypísať názov a absolútnu cestu k súboru, veľkosť a počet fragmentov
* prijať súbor od vysielača
* poslať správu
* ukončiť spojenie

### Vysielač

* Nastaviť si port na počúvanie
* Nastaviť miesto kam sa súbor uloží
* Potvrdiť prijatie rámcov
* Pospájať fragmenty
* Vykonať výmenu s klientom
* Detegovať chybu a vyžiadať znovu poslanie rámca
* Oznámiť prijatie nesprávnej správy
* Oznámiť poradie prijatého fragmentu
* Uložiť súbor, vypísať kam sa uložil a počet fragmentov

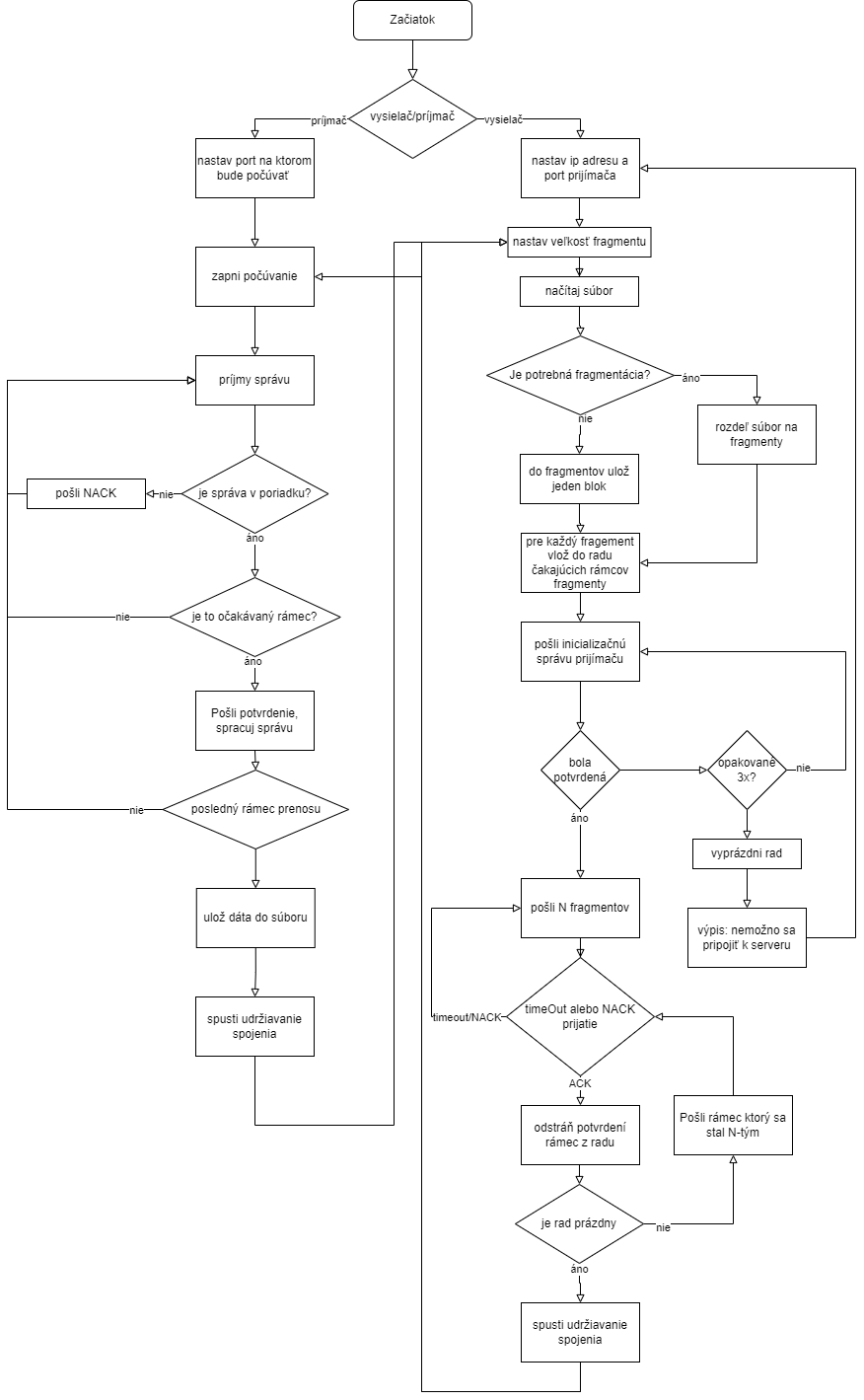
### Štruktúry

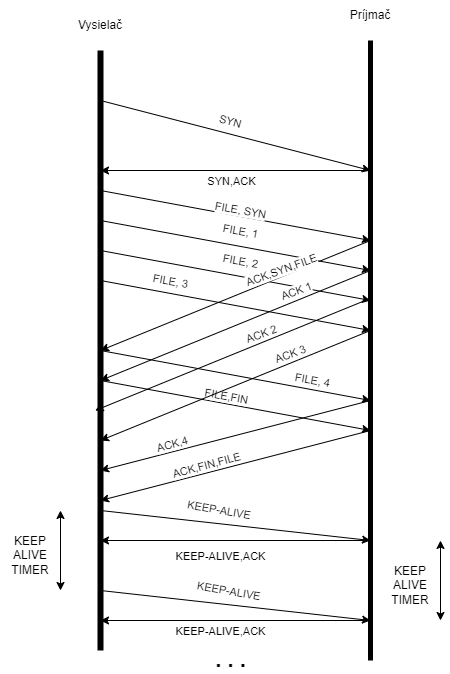
* Spojenie – všetky potrebné údaje na posielanie a získavanie dát
* Konštanty – všetky konštanty ako sú flag-y, typy

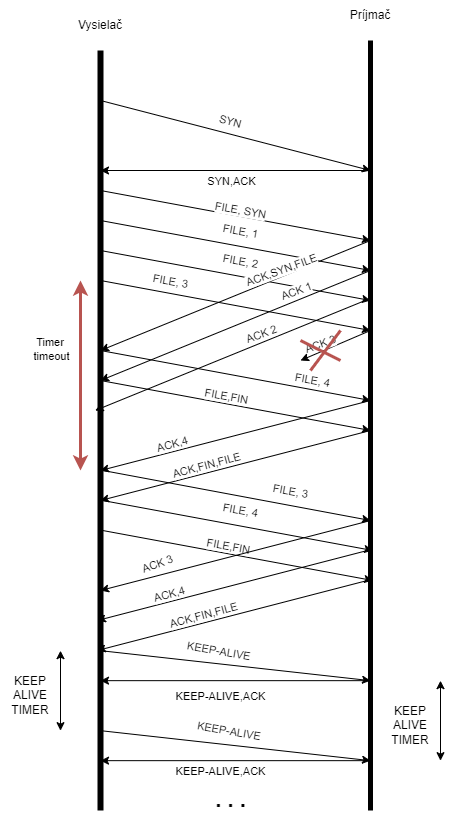
### Thready

Program sa bude skladať z troch vlákien a jedného hlavného, v ktorom budeme brať požiadavky od používateľa. Jedno vlákno bude použité na prijímanie a druhé na posielanie rámcov. Tretie vlákno bude slúžiť na posielanie KEEP-ALIVE rámcov a aj na zistenie či prišiel KEEP-ALIVE rámec do určitého času.

## Diagram fungovania







# Implementácia

## Rozdiely od návrhu

**Hlavička**

V hlavičke sme zväčšili pole FRAG\_COUNT na 3 Bajty, kvôli podpore prenosu 2MB súboru po 1 bajte. Zároveň sa pole FRAG\_COUNT nepoužíva len na prenos informácie o počte fragmentov súboru, ale aj na zistenie poradia fragmentu.

Pole SEQ\_NUM je pole pre určenie, obsahuje sekvenčné číslo rámca, aby arq metóda vedela, ktorý rámec očakáva, ktoré prišli skôr a zároveň či prišiel predchádzajúci rámec. Táto hodnota sa cyklí, teda ak posledný prijatý rámec bol 65535 tak ďalší bude 0.

0 7 15 31

|  |  |  |
| --- | --- | --- |
| TYPE | FLAG | SIZE |
| SEQ\_NUM | | FRAG\_COUNT |
| FRAG\_COUNT |  | |
| DATA | | |
| CHECKSUM | |

**Kontrola chýb**

Na kontrolu chýb sme použili CRC16 presnejšie implementáciu XMODEM zo stránky <https://mdfs.net/Info/Comp/Comms/CRC16.htm> a upravili ju na verziu, ktorá pôjde spustiť v jazyku python.

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok, na ktorom je text

Automaticky generovaný popis

**Fragmentácia**

Vďaka zväčšeniu FRAG\_COUNT, nemusíme overovať, či veľkosť zadaného fragmentu je dostačujúca na prenos súboru. Ďalej sme kvôli odporúčanému limitu 576 Bajtov na UDP payload rozhodli, že maximálna veľkosť na prenos dát bude 565 Bajtov (hlavička + crc = 11B). Teda presne sa môže preniesť maximálne 576 Bajtov. Ak používateľ zadá veľkosť fragmentu viac ako 565, tak sa veľkosť automaticky nastaví na 565 Bajtov.

**Udržiavanie spojenia**

Na rozdiel od návrhu, klient ani server neposielajú ukončovacie rámce, resp. na ukončenie spojenia je potrebné jednoducho nechať vypršať čas čakania na keep-alive, či už pre server alebo klient. Ukončovací rámec sa posiela iba vtedy, keď sa strany vymenia a posielajúca strana už nechce posielať ďalšie údaje prijímaču.

**Sekvenčný diagram**

Sekvenčný diagram funguje stále rovnako, jediným rozdielom je, že FILE | SYN rrámec môže byť rozdelený na viac fragmentov podľa dĺžky názvu a veľkosti fragmentu. Ďalej sa už neposiela FILE | FIN a FILE | FIN | ACK, prenos súboru je potvrdený a dokončený získaním všetkých rámcov súboru.

**Knižnice**

Pribudli knižnice:

* Random – pre simulovanie chyby
* Os – kontrola existencie priečinka a výpis absolútnej cesty
* Re – overenie správnosti zadanej IP adresy

## Funkcionalita

**Typy sprav pri prenose**

1. **CONTROL | SYN –** posiela klient serveru pri inicializácií komunikácie,
2. **CONTROL | SYN | ACK –** server potvrdzuje,
3. **CONTOL | FIN –** ukončenie spojenia,
4. **TXT –** rámce textu,
5. **FILE | SYN –** prenášanie názvu súboru,
6. **FILE –** rámce súboru,
7. **CONTROL | ACK –** odpoveď na prijatie rámcov,
8. **CONTROL | SWAP –** vyžiadanie výmeny strán,
9. **CONTROL | SWAP | ACK –** potvrdenie odpovede na výmenu,
10. **CONTROL | KEEP –** rámec na udržanie spojenia,
11. **CONTROL | KEEP | ACK –** potvrdenie udržania spojenia.

Pri rámcoch TXT, FILE a FILE | SYN je prvým prenášaným rámcom prenášane nie dáta, ale informácia o počte posielaných rámcov textu.

**Simulovanie chyby**

Ak sa používateľ rozhodne simulovať chybu pri prenose, tak každý desiaty dátový rámec prenosu bude pozmenený pri jeho prvotnom odoslaní a následne pri znovu posielaní okien bude už rámec v poriadku. Ak neprenášame viac ako 10 dátových rámcov, tak sa označí ako chybný posledný prenášaný dátový rámec.

# Použitie

## Spustenie

Na spustenie je potrebne použiť **python 3.10.0 64-bit** pomocou príkazu py main.py v príkazovom riadku

## Používateľské rozhranie

**Všeobecné**

Ako prvé pri spustení programu sa používateľa spýta kam chce prijaté dáta ukladať. Ak používateľ nič nezadá, tak sa súbor automaticky ukladá do priečinka Downloads v pracovnom adresári. Ak zadá neplatný priečinok, tak program od neho bude požadovať znovu zadanie priečinku.



Po zadaní priečinku sa používateľa spýta, ktorý mód si chce prvotne zapnúť. 1 – server, 2 – klient a 0 – vypne program. Každá iná možnosť požiada používateľa o opravu.

Obrázok, na ktorom je text

Automaticky generovaný popis

**Server**

Ak si zvolíme server, tak nám da na výber rozhrania, na ktorých môžeme počúvať. Pre výber zadáme číslo rozhrania. Ak zadáme zle údaje, tak nás program požiada znovu o zadanie korektného rozhrania. Po výbere rozhrania si používateľ nastaví port aký chce používať. Na výber má od 1024 po 65535.

**Obrázok, na ktorom je text

Automaticky generovaný popis** **Obrázok, na ktorom je text

Automaticky generovaný popis**

**Obrázok, na ktorom je text

Automaticky generovaný popis**

Po tejto inicializácii sa vypíšu údaje o Serveri a o tom, že boli zapnuté vlákna na posielanie a počúvanie. Síce boli aj vypísané akcie na poslanie správy a súboru, ale ich zadanie vypíše v tomto štádiu iba . Pre ich použitie sa musí klient pripojiť na server a požiadať o výmenu.

Obrázok, na ktorom je text

Automaticky generovaný popis

**Klient**

Pri výbere klienta nás program požiada o zadanie IP adresy servera a portu, na ktorom počúva. Po zadaní informácií sa vypíšu informácie o klientovi a to na aký server sa pripája a na aký port.



Obrázok, na ktorom je text

Automaticky generovaný popis

**Posielanie**

Poslanie textovej správy. Pri poslaní textovej správy program povoľuje posielanie iba ASCII znakov. Pred poslaním sa požiadame používateľa o veľkosť fragmentu aký chceme poslať. Následne o zadanie správy a nakoniec či si používateľ praje simulovať chybu. Po zadaní informácií sa vypíše veľkosť a počet fragmentov správy a začnú sa posielať dáta

**Obrázok, na ktorom je text

Automaticky generovaný popis**

Pri prijatí na serveri sa vypisujú aký rámec prijal a, že či to je očakávaný rámec. Prvým prijatým rámcom je rámec na začatie komunikácie a server vypíše  po prijatí všetkých rámcov textu vypíše veľkosť prenesených údajov, počet fragmentov a samotnú správu

**Obrázok, na ktorom je text

Automaticky generovaný popis**

Pri posielaní súboru sa používateľa spýtame na veľkosť fragmentu a názov súboru, ktorý chceme poslať. Po zadaní sa vypíše absolútna cesta k súboru, na koľko častí bol rozdelený názov aj dáta a aj veľkosť názvu a dát. Keď sa dáta posielajú, tak vypisujeme progress bar, ktorý ukazuje na koľko percent sme už preniesli dáta. Prvý je na zobrazenie prenosu názvu a druhý na prenos dát súboru.

Obrázok, na ktorom je text, elektronika, batéria

Automaticky generovaný popis

Po prijatí súboru serverom sa napíše že súbor bol prijatý, vypíše miesto kde sa uložil a aj počet prijatých fragmentov a veľkosť súboru. Informácie o názve súboru sa vypisujú ešte pred samotným prijímaním dát.

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok, na ktorom je text

Automaticky generovaný popis

Pri posielaní keep alive vysielajúca strana vypíše  keď posiela keep alive a  keď dostane odpoveď. Server vypíše, že udržiava spojenie s adresou klienta.



Ak sa stane, že pri odosielaní prijímajúca strana nebude odpovedať, tak sa vypíše, že nastal timeout a okno sa znovu pošle. Rovnako aj pri odpovedi NACK tak vypíšeme, že bol prijatý a posielame okno znovu. Pri timeout-e, ak server neodpovie do 3 ďalších pokusov, tak sa klient od servera odpojí a buď používateľ znovu skúsi poslať inicializačný rámec a aj dáta alebo sa vypne 0 a znovu zapne a zadá dobre údaje.

Obrázok, na ktorom je text

Automaticky generovaný popis



Pri výmene sa vypíše na strane prijímača, že teraz môže posielať dáta on a na strane klienta sa vypíše, že druhá strana potvrdila výmenu a prepne sa do režimu počúvania.



# Príklad komunikácií

Na obrázku je zobrazený prenos dát, kde každý druhý rámec bol prvotne odoslaný ako chybný.

Obrázok, na ktorom je stôl

Automaticky generovaný popis

Ack starých rámcov

ďalej

Ignore out of Order

Ignore out of Order

NACK

NACK

NACK

rePoslanie

okna

rePoslanie

okna

rePoslanie

okna

Poslanie

okna

Init

Obrázok, na ktorom je stôl

Automaticky generovaný popis

Keep alive timeout

Keep alive

Keep alive

Keep alive

Keep alive