Jozef Varga, ID: 86060

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Komunikácia s využitím UDP protokolu

Počítačové a komunikačné siete

Jozef Varga
Dr. Ing. Michal Ries
2017/2018

Obsah

Jozef Varga, ID: 86060

Zadanie

Nad protokolom UDP (User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP navrhnite a implementujte program, ktorý umožní komunikáciu dvoch účastníkov v sieti Ethernet, teda prenos správ ľubovoľnej dĺžky medzi počítačmi (uzlami).

Program bude pozostávať z dvoch častí – vysielacej a prijímacej. Vysielací uzol pošle správu inému uzlu v sieti. Predpokladá sa, že v sieti dochádza k stratám dát. Vysielajúca strana rozloží správu na menšie časti - fragmenty, ktoré samostatne pošle. Správa sa fragmentuje iba v prípade, ak je dlhšia ako max. veľkosť fragmentu. Veľkosť fragmentu musí mať používateľ možnosť nastaviť menšiu ako je max. prípustná pre linkovú vrstvu.

Po prijatí správy na cieľovom uzle tento správu zobrazí. Ak je správa poslaná ako postupnosť fragmentov, najprv tieto fragmenty spojí a zobrazí pôvodnú správu. Komunikátor musí vedieť usporiadať správy do správneho poradia, musí obsahovať kontrolu proti chybám pri komunikácii a znovuvyžiadanie rámca, vrátane pozitívneho/negatívneho potvrdenia. Pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia. Odporúčame riešiť cez vlastne definované signalizačné správy.
Program musí byť organizovaný tak, aby oba komunikujúce uzly mohli byť (nie súčasne) vysielačom a prijímačom správ.

Program musí mať nasledovné vlastnosti (prvých 5 minimálne):

1. Program musí byť implementovaný v jazyku C/C++ s využitím definovaných knižníc (schválených cvičiacim) a skompilovateľný a spustiteľný na PC.
2. Pri posielaní správy musí používateľovi umožniť určiť cieľovú stanicu.
3. Používateľ musí mať možnosť zvoliť si max. veľkosť fragmentu.
4. Obe komunikujúce strany musia byť schopné zobrazovať:
    a. poslanú resp. prijatú správu,
    b. veľkosť fragmentov správy.
5. Možnosť odoslať chybný rámec (do rámca je cielene vnesená chyba, to znamená, že prijímajúca strana zdeteguje chybu pri prenose).
6. Možnosť odoslať dáta zo súboru a v tom prípade ich uložiť na prijímacej strane do súboru.

Jozef Varga, ID: 86060

Analýza

Sieťová komunikácia je pomerne široký pojem v informatike. Vzhľadom na jej široku problematiku existuju modely, ktoré sa nám ju snažia jednoduchšie vysvetliť. Každý model sa skladá z vrstiev, ktoré majú určitú úlohu (buď HW alebo SW). Jedným z týchto modelov je aj model TCP/IP ,ktorý sa skladá z týchto vrstiev:

- Aplikačná vrstva
- Transportná vrstva
- Sieťová vrstva
- Vrstva sieťového rozhrania

Každá vrstva má presne definovaný spôsob komunikácie so susednou nižšou / vyššou vrstvou. Nižšia vrstva vždy poskytuje službu vyžšej vrstve. Pri komunikácii v sieti spolu komunikujú rovnocenné vrstvy.

Aplikačná vrstva nám slúži na prenos dát. Poskytuje používateľom sieťové služby (rozhranie k sieťovým službám), stará sa o reprezentáciu dát, kompresiu, šifrovanie (preloží data z obyčajného formátu do formátu, ktorému rozumie táto vrstva), taktiež nadväzuje, udržiava a ruší logické spojenia medzi užívateľmi. Táto vrstva môže zabezpečovať aj akúsi kontrolu pri prenose (napr. môže znovu vyžiadať data), to sa využíva najčastejšie pri menej spoľahlivých protokoloch ako je UDP.

Najznámejšie protokoly aplikačnej vrstvy :

- HTTP – protokol na prenos hypertextových dokumentov (nezabezpečený)
- HTTPS – to iste ako HTTP no je zabezpečený (bankové transakcie)
- FTP – slúži na prenos dát/ súborov
- DNS - mapuje IP adresu na dománové meno
- DHCP - prideluje základnú konfiguráciu
- Telnet – nezabezpečený vzdialený prístup na iné zariadenie
- SSH- zabezpečený vzdialený pristup na iné zariadenie
- TFTP – jednoduchý protokol na prenos súborov (zavádzanie systámu do routra)

Transportná vrstva zabezpečuje tok dát medzi dvoma komunikačnými uzlami. Využívajú sa tu zvyčajne tieto dva protokoly:

- UDP protokol – je to protokol bez spojenia a bez potvrdenia prijatia odoslaných dát. Prenášané data sa volajú datagramy. Pri strate dát alebo chybnom prenose sa o tento problém stará aplikačná vrstva a jej protokoly.
- TCP protokol – je spojovo orientovaný protokol ktorý pri začiatku komunikácie vykoná takzvaný three-way-handshake (trojcestné podanie ruky) na synchronizáciu údajov potrebných na bezpečné posielanie správ. Tento protokol zabezpečuje preposielanie chybných segmentov, riadi celú komunikáciu a zabezpečuje celkovú kvalitu služieb.

Rozdiel medzi týmito dvoma protokolmi je hlavne v rýchlosti. Programátor / aplikácia si zvolí, ktorý protokol využíva nakoľko v niektorých prípadoch nám ide iba o rýchlosť prenosu a dokážeme zanedbať malé straty (napr. hranie online hry). Na tejto vrstve sú zariadenia adresované pomocou portov. Oba protokoly majú zdrojový (z akého portu to bolo poslané)

a cieľový port (na aký port to má prísť príjmacému komunikačnému uzlu). Na týchto portoch čakajú jednotlivé služby (protokoly) a počúvajú, či im neprišli nejaké data. Ak na počítači robíme viacero vecí a preto je aktívnych viacero portov (viacero protokolov) využívame takzvané multiplexovanie (prepínanie dát) – dáta sa delia do im pridelených portov.

Sieťová vrstva prenáša takzvané pakety. Zabezpečuje prenos paketov od zdroja cez sieť až ku cieľu (hľadá najlepšiu cestu). Smeruje a prepína pakety na tejto ceste pomocou smerovacích protokolov. Stará sa o fragmentáciu dát a prenáša ich koncovému užívateľovi. Na sieťovej vrstve sú komunikujúce zariadenia adresované pomocou IP adries a ich masiek.
Najznámejšie protokoly sieťovej vrstvy:

- Internet Protocol (IP) – nesie pakety bez spojenia, potvrdenia do cieľa. Stará sa o chod siete (ak sa niečo stratí, nevadí, sieť funguje ďalej). Primárnou úlohou tohto protokolu je čo najrýchlejšie doručiť dáta. O stratené/ chybné dáta sa v tomto prípade starajú vyššie vrstvy.
- ICMP – slúži na overenie konektivity cieľa. Je to takzvaný ping.

Vrstva sieťového rozhrania prenáša bity a rámce. Rieši parametre prenosového média, fyzické rozhranie, priemyselné štandardy, HW a iné. Vrstva nerieši, aký HW sa má použiť ale využíva existujúce prenosové technológie. Zariadenia sú adresované pomocou MAC adries (zdrojová a cieľová MAC adresa). MAC adresu má každý počítač unikátnu, narozdiel napríklad od IP adresy. Toto adresovanie sa využíva pri hľadaní cieľa v lokálnej sieti.
Najznámejssie protokoly:

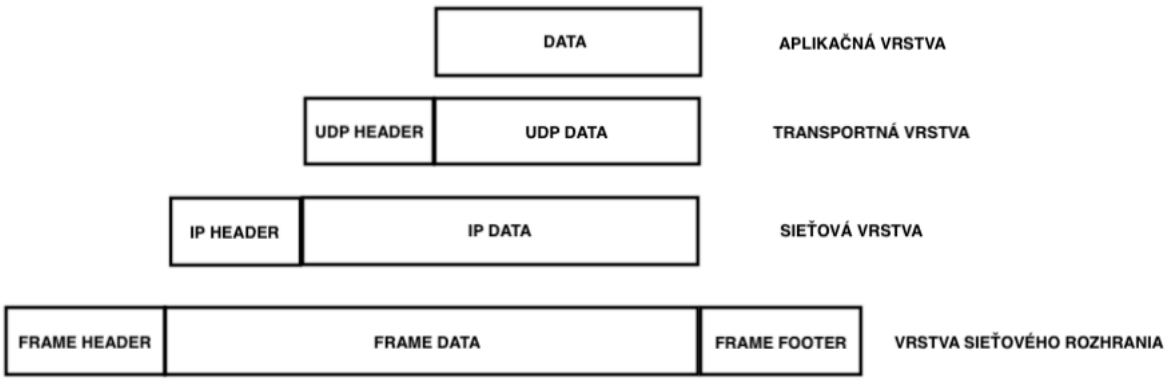
- ARP k danej IP adrese vyhľadá MAC adresu stroja v lokálnej sieti (opak RARP)
- PPP komunikačný protokol pre priame spojenie medzi uzlami (umožnuje autentifikáciu, šifrovanie a kompresiu)

Enkapsulácia a hlavičky

Enkapsulácia spočíva vo vložení PDU (Protocol Data Unit) z vyššej vrstvy do protokolovej jednotky nižšej vrstvy. To zabezpečuje aby vyššia vrstva mohla používať služby nižšej vrstvy a aby data mohli byt prepravené sieťou k rovnakej vrstve cieľového uzla. Niekedy nižšia vrstva môže rozdelit PDU vyšších vrstiev do niekoľkých svojich PDU (segmentácia, fragmentácia) alebo ich môže naopak spojiť. Pri prijatí dát (ideme TCP/IP modelom nahor) každá vrstva skontroluje svoju hlavičku a ak PDU nie je určený pre ňu, zahadzuje ho.

Jozef Varga, ID: 86060

Pri modeli TCP/IP (UDP protokol) funguje enkapsulacia takto:



Obrázok 1 Enkapsulácia UDP protokol

UDP na transportnej vrstve pridá UDP hlavičku

UDP Header	
Zdrojový port	Cieľový port
Dĺžka	Checksum

Najpodstatnejšie polia v hlavičke sú pre nás cieľový a zdrojový port. Tieto polia nám hovoria o protokole ktorý využívame (ak sú to porty zo známych protokolov) a o tom na akom porte bude cieľová stanica očakávať našu správu, ktorú pošleme zo zdrojového portu. Po pridaní UDP hlavičky vytvoríme takzvaný datagram (DATA + UDP hlavička) posunieme nižšej vrstve (sieťovej) ktorá jej pridá takzvanú IP hlavičku.

IP Header				
Verzia	IHL	Druh služby	Celková dĺžka bytov v pakete	
Identifikácia			Flag	Fragment offset
Time to live		Protocol	Header checksum	
Zdrojová IP adresa				
Cieľová IP adresa				
Options				

Jej najpodstatnejšie polia sú verzia (hovorí nám to o verzii IP protokolu), IHL (dĺžka IP hlavičky), Protocol (druh protokolu – teraz to bude UDP keďže nám došiel datagram), zdrojova IP adresa (moja IP adresa), cieľová IP adresa (adresa zariadenia na ktoré zasielam správu). IP hlavička sa pridáva pred UDP hlavičku (pred datagram) a tým sa vytvorí výsledný paket. Takto vytvorený paket posunieme nižšej vrstve (vrstve sieťového prístupu) ktorá pred paket (pred IP hlavičku) pridá svoju vlastnú Frame hlavička.

Frame Header		
Cieľová MAC adresa	Zdrojová MAC adresa	EtherTyp

Frame hlavička obsahuje Cieľovú MAC adresu ako prvú (pre rýchlejšie prepínanie) tá obsahuje MAC adresu nasledujúceho zariadenia. Ďalej obsahuje zdrojovú MAC adresu (adresu zariadenia, ktoré to posiela, tu je podstatné si uvedomiť že MAC adresa sa môže počas prenosu dát meniť) a ako posledné EtherTyp, ktorú určuje typ technológie využitej pri zasielaní dát (Ethernet 2, 802.2 LLC ...). Pri vytváraní frame-u sa pridá aj takzvaná preambula (signalizuje začiatok správy) a pri tejto vrstve sa pridá aj päta, ktorá obsahuje FCS (kontrolný súčet frame-u).

Hlavné vlastnosti protokolu UDP z pohľadu zadania

UDP protokol je nespoľahlivý, nespojitý protokol, ktorý nezasiela potvrdenie o prijatí správ. Podporuje multicast a broadcast vysielanie. Po pridaní UDP hlavičky vytvára takzvaný UDP datagram, ktorý je narozdiel od TCP (TCP je spoľahlivý protokol) menší (hlavička obsahuje menej informácií) a tým, že nevyužíva overenie (potvrdenie) je rýchlejší. Tým že nezasiela potvrdenie o prijatí správy (alebo potvrdenie či správa prišla v poriadku) tak UDP nedetekuje straty dát (negarantuje doručenie dát). A ani nezaručuje že dáta prídu v rovnakom poradí ako boli vyslané. UDP nemá mechanizmus, ktorý by riadil tok dát a tým môže dochádzať ku zahlteniu. Aby sa sieť nezahlcovala, v sieti prebieha zahadzovanie paketov sieťovými zariadeniami. Hlavičku UDP tvorí cieľový/ zdrojový port, dĺžka a kontrolný súčet. Tým že UDP neposiela potvrdenie, často sa zdrojový port nastaví na nulu (ak sa nepoužíva). Taktiež kontrolný súčet môže byť vynechaný. Spoľahlivosť ale môže zaručiť aplikačná vrstva napr. TFTP protokol využíva jednoduchý mechanizmus, ktorý pridáva UDP protokolu spoľahlivosť.

Chyby v prenose a znovuvyžiadanie rámca

UDP protokol nevyužíva znovuvyžiadanie dát a ani nepotvrdzuje správnosť potvrdenia. O chyby v prenose sa stará vyššia vrstva ktorá využíva kontrolný súčet (CRC). Chyba pri prenose môže nastať z viacerých dôvodov napríklad pri prenose metalickým káblom nastane nejaké rušenie, alebo nejaké nečistoty v optickom vlákne. Chyba sa väčšinov prejaví zmenou niektorých bitov na opačné. Pri odoslaní sa vypočíta CRC a uloží sa do hlavičky. Pri prijatí sa CRC z hlavičky porovnáva z CRC vypočítaným z prijatého rámca.

Veľkosť fragmentu

Používateľ na začiatku programu musí nastaviť veľkosť fragmentu podľa ktorého sa budú následne dané správy rozdelovať na menšie fragmenty a až tak zasielať. Veľkosť fragmentu musí byt väčšia ako 20 bytov a menšia ako 65535. Server fragmenty príjme a následne spojí (ak nastane chyba vyžiada preposlanie).

Jozef Varga, ID: 86060

Špecifikácia požiadaviek

Používatľ má možnosť:

1. výberu spustenia aplikácie v móde Klienta/ Servera
2. nastaviť IP adresy servera (mód klient)
3. nastaviť port na ktorom počúva server (mód klient)
4. nastaviť veľkosť fragmentu (mód klient)
5. nastaviť port na ktorom má server počúvať (mód server)
6. poslať správu
7. poslať správu s chybným fragmentom
8. pozrieť si predchádzajúce odoslané / pijaté správy

Ostatné požiadavky:

1. aplikácia využíva CRC (kontrolný súčet)
2. klientsky mod rozdelí v prípade potreby správu na menšie fragmenty
3. server aj klient má možnosť zobrazit prijatú/odoslanú správu a veľkosť jej fragmentov
4. server usporiada prijaté fragmenty do správneho poradia a vypíše správu
5. server zdetekuje a následne si vyžiada preposlanie chybnej správy
6. server potvrdí prijatie správy
7. klientska aplikácia vysiela keep alive spravy na server pri nečinnosti
8. aplikácia je naprogramovaná v jazyku C/C++

Knižnice:

1. sys/socket.h
2. arpa/inet.h
3. netinet/in.h

Jozef Varga, ID: 86060

Návrh riešenia

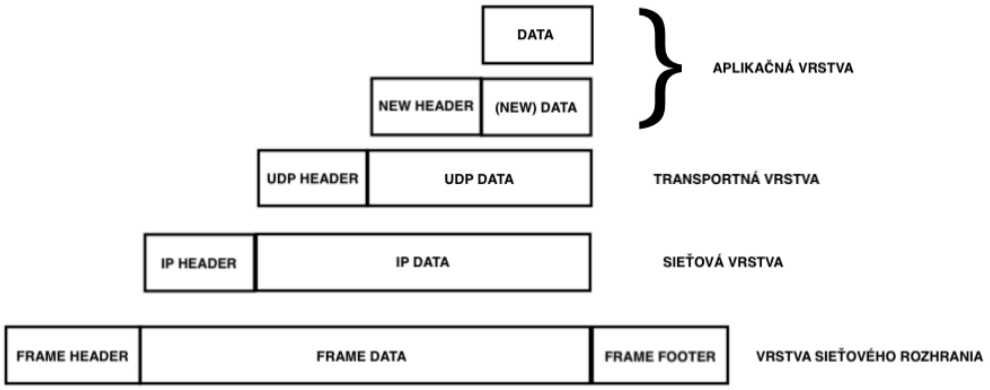
Návrh vlastného protokolu

Náš protokol bude využívať vlastnú hlavičku. Táto hlavička bude pozostávať s 5 buniek a to :

1. typ správy ktorý určuje význam správy.
    a. 0 určuje inicializáciu pred zaslaním dát a oznamuje správne doručenie dát
    b. 1 určuje zaslanie/prijatie dát
    c. 4 slúži na znovuvyžiadanie dát serverom
    d. 5 určuje zaslanie/prijatie JPG súboru
    e. 6 určuje zaslanie/prijatie PNG súboru
    f. 7 určuje zaslanie/prijatie TXT súboru
    g. 8 určuje zaslanie/prijatie PDF súboru
    h. 9 určuje zaslanie/prijatie PPT súboru
    i. 10 určuje zaslanie/prijatie C súboru
    j. 11 určuje zaslanie/prijatie DOCX súboru
    k. 22 určuje keep alive paket
2. Poradové číslo slúži na znovuzoskupenie správy (zoradenie fragmentov do správneho poradia) a pri inicializácii správy obsahuje celkovú veľkosť správy
3. Veľkosť fragmentu je aktuálna veľkosť fragmentu
4. Počet fragmentov nám hovorý o celkovom počte fragmentov (rozdelenia správy)
5. CRC je kontrolný súčet

Obrázok 2 Hlavička nášho protokolu

Našu hlavičku budeme sledovať/ rozbaľovať/ spracovávať v našom programe teda na aplikačnej vrstve.



Obrázok 3 Náš protokol pri enkapsulácii (NEW HEADER)

Pri každej odoslanej správe (text/ súbor) zašleme inicializačný frame ktorý bude obsahovať informácie o celkovej správe. Teda ak by sme posielali text naša počiatočná inicializačná správa odoslaná klientom by vyzerala následovne:

1. Typ správy = 0 (nakoľko to signalizuje inicializáciu)
2. Poradové číslo by obsahoval veľkosť celej textovej správy pre jednoduchosť napríklad 10. Táto bunka inač indikuje poradie fragmentu ale pri inicializácii má informatívny význam pre server.
3. Veľkosť fragmentu obsahuje nami zadanú hodnotu na začiatku maximálnej veľkosti fragmentu pri odoslanej správe aby server vedel akú najväčšiu správu môže dostať.
4. Počet fragmentov obsahuje počet na koľko fragmentov sme danú správu rozdelili.
5. CRC je vypočítane z inicializačnej správy aby server dokázal zistiť či mu daná inicializačná správa prišla v poriadku.

Klient po odoslaní tejto správy čaká na odpoveď servera.
Server správu príjme vypočíta vlastné CRC a následne ho porovná z CRC v našej hlavičke. Ak sú rovnaké načíta si informácie z hlavičky. Server celý čas načúval na porte ktorý sme mu zadali a prímal data o veľkosti našej hlavičky. Ak všetky údaje načíta odošle odpoved ktorá obsahuje presne tie isté údaje ako inicializačná správa (prepošle ju). Ak by mu nesedel kontrolný súčet taktiež prepošle tú istú správu klientovy akurád zmení typ správy na 4 (žiadosť o preposlanie dát).
Ak klient príme zápornú odpoveď len prepošle danú správu, ak príde kladná odpoveď rozfragmentuje textovú správu a každému fragmentu vytvorý hlavičku ktorá obsahuje:

1. Typ správy ktorý závisí na tyu dát ak je do text bude to 1 ak to je neaký typ súboru bude to jedna z hore uvedených značiek.
2. Poradové číslo teraz už obsahuje reálne poradové číslo fragmentu (ak je to 56. kúsok dát bude obsahovať číslo 55 – číslujeme od 0)
3. Veľkosť fragmentu obsahuje reálnu veľkosť fragmentu
4. Počet fragmentov je nemenné pole to ostáva počas celého posielania správy rovnaké
5. CRC sa bude pri každom fragmente prepočítavať.

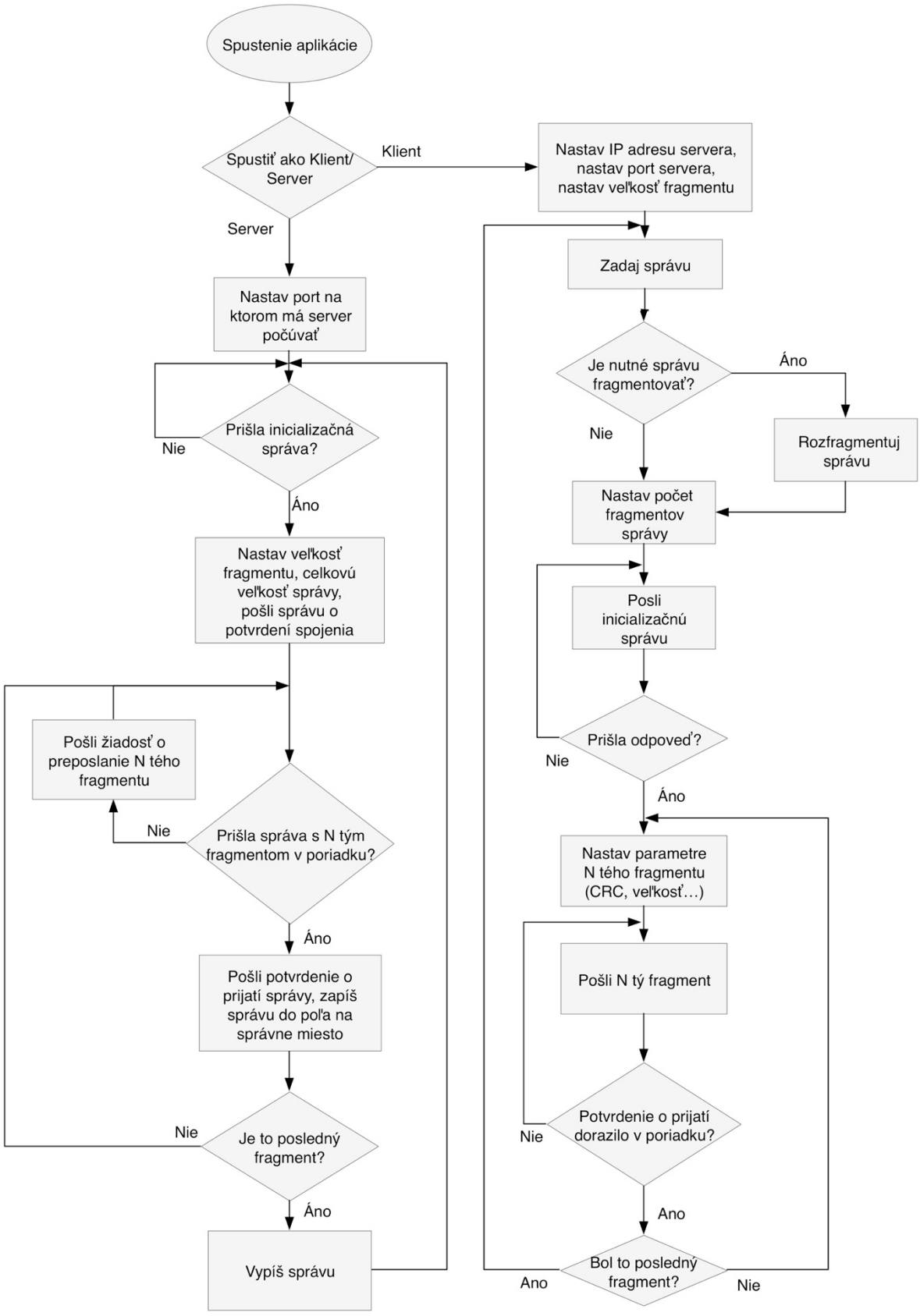
Fragmenty klient začne po jednom posielať a následne vyčkávať na odpoveď od servera. Server fragmenty príme. Vypočíta a porovná CRC, ak nesedia zašle správu s typom 4. Ak je všetko v poriadku príme data a uloží ich do pola. Pri ukladaní mu pomáha poradové číslo fragmentu, veľkosť fragmentu (nastavená užívateľom ktorá bola zaslaná v inicializačnej správe) a veľkosť prijatého fragmentu. Vďaka tomu si v poli vytvorí kópiu správy ktorú po prijatí všetkých fragmentov vypíše na obrazovku. Ak by to nebol text ale napríklad PDF proces je identický akurád nezapisuje do poľa ale do súboru rovnakého typu. Pri zasielaní súboru si definujeme maximálnu veľkosť súboru na 100MB (realne je program ale obmedzený veľkosťou voľnej RAM) a na text je obmedzenie 5000 znakov v jednej správe (toto obmedzenie sa môže líšiť podľa kompilátora napr. v Xcode je to 1024 znakov – to je obmedzenie kompilátora).

CRC (kontrola cyklickým kódom) slúži na overenie správnosti údajov (či sa pri posielaní v správe niečo nezmenilo). Táto kontorla prebehne pred odoslaním (hodnota sa zapíše do hlavičky) a po prijatí (hodnota sa sa vypočíta na novo a porovnáva sa s hodnotou v hlavičke). Je to v podstate algoritmus ktorý každý retazec zašifruje na unikátne číslo. Môj výpočet realizujem takto

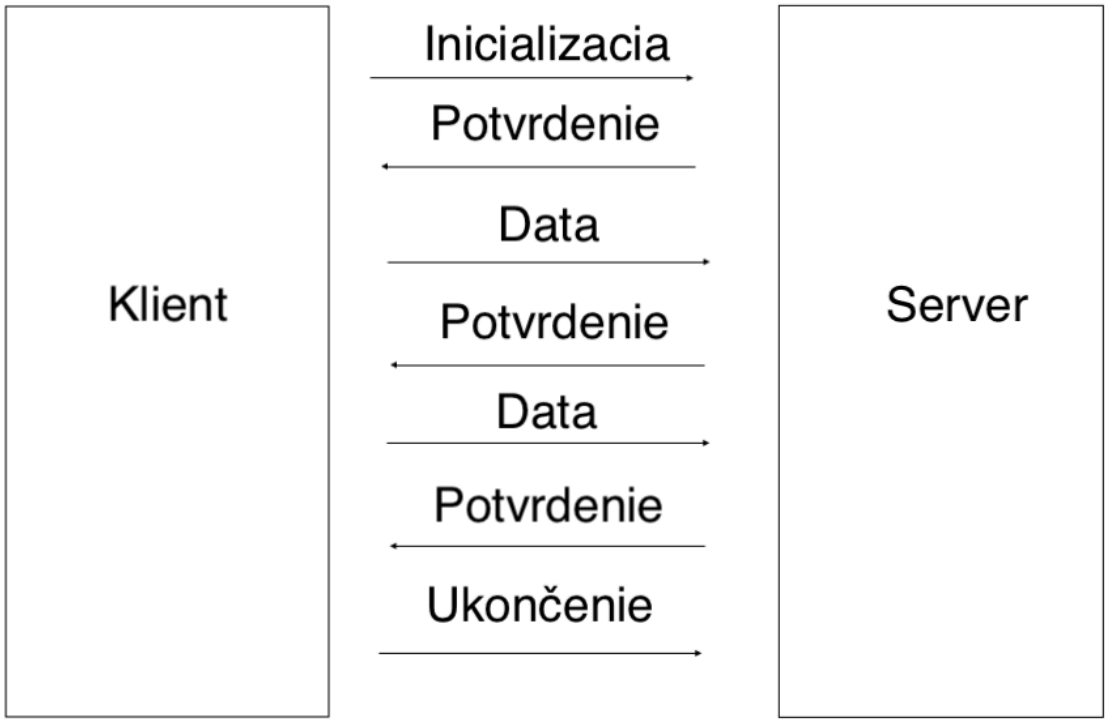
```
1   void funkcia Vypocet_CRC(ukazovatel_na_zaciatok_dat, dlzka_spravy){
2       CRC <- 1;
3       for i from 0 to (dlzka_spravy - 1){
4           pomoc := ukazovatel_na_zaciatok_dat;
5           for x from 0 to 7 {
6               if (CRC and 0x8000) = 1 {
7                   CRC := (CRC leftShift) or (pomoc and 1);
8                   CRC := CRC xor 0x1083;
9               }
10              else
11                  CRC := (CRC leftShift) or (pomoc and 1);
12              pomoc := pomoc rightSift;
13          }
14      }
15      return CRC;
16  }
```

Obrázok 4 Pseudokód CRC16

Vývojový diagram



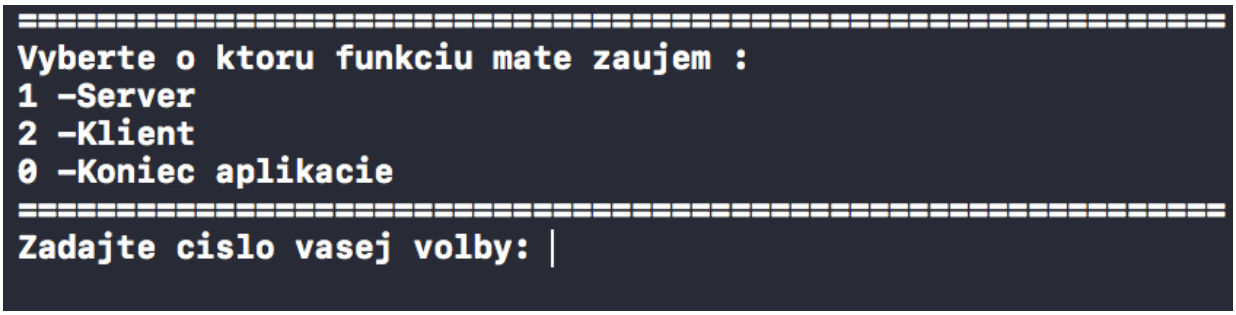
Obrázok 5 Diagram Klient/ Server aplikácie



Obrázok 6 Komunikácia

Používateľské rozhranie

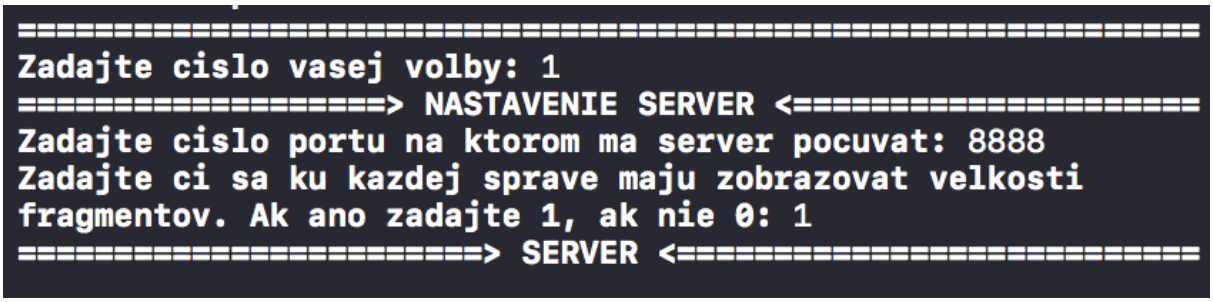
Po štarte aplikácie sa užívateľovy zobrazí nasledujúce menu:



Obrázok 7 Zapnutie aplikácie (výber funkcionality)

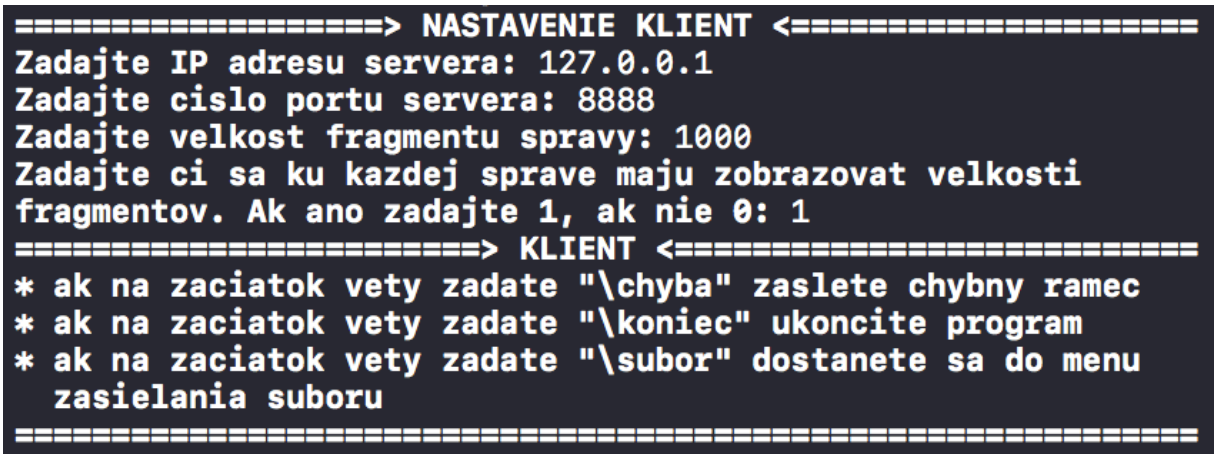
Užívateľ má možnosť vybrať si či aplikáciu chce spustit ako Klient - 2 (zasiela správy na server), alebo ako Server - 1(príma správy od Klienta). Treťou možnosťou je ukončiť aplikáciu - 0.

Ak aplikáciu užívateľ spustí ako Server (1) dostane možnosť nastaviť port na ktorom server čaká na správu a či chcete aby zobrazovalo aj veľkosť prenesených fragmentov správy . Ak vami nastavený port už je obsadený, aplikácia oznámy chybu a ukončí sa.



Obrázok 8 Nastavenie Servera

Pri výbere Klienta (2) sa vypíše aj menu ktoré informuje o možnosti zasielaných správ



Obrázok 9 Nastavenie Klienta

Jozef Varga, ID: 86060

Implementácia

Zmeny oproti návrhu

Tým že som si naštudoval pomerne veľa literatúry, podarilo sa mi zostaviť návrh pomerne presný a vďaka nemu bola implementácia totožná s návrhom. Jediná zmena bola využitie viacerých knižníc, napríklad pthread.h, ktorá nám zabezpečuje prácu s vláknami, ktoré využívam najmä pri keep alive pakete a pri indikácii odpojenia jednej zo strán..

Použité knižnice a funkcie

Knižnice:

#include<stdio.h> - základná knižnica
#include<string.h> - práca so stringom (memset, malloc...)
#include<stdlib.h> - základná knižnica
#include<arpa/inet.h> - práca s IP adresov
#include<sys/socket.h> - práca so socketom (otvorenie/zatvorenie spojenia)
#include<pthread.h> - práca s vláknami.

Štruktúry:

struct hlavicka_fragmentu{} – obsahuje prvky hlavičky protokolu, ktorý sme si navrhli.
struct argument_vlakna{} – štruktúra, ktorá nám slúži ako argument do vlákna. Obsahuje informácie o sockete.

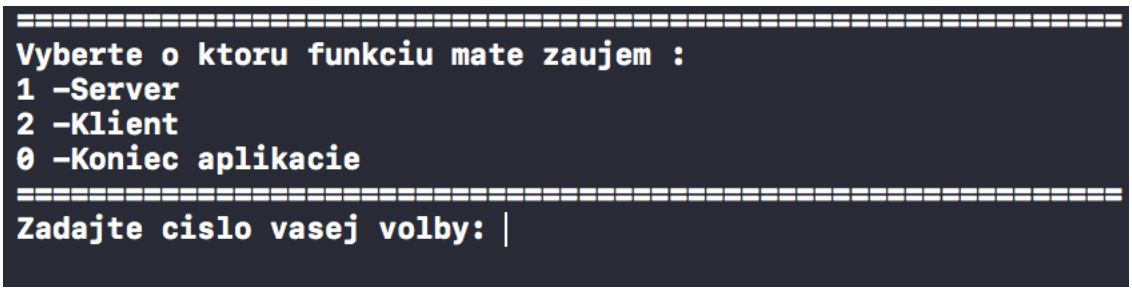
Funkcie:

void *vlakno_chyba_SERVER(void *vargq){} – Slúži na kontrolovanie servera (na ukončenie servera počas behu, aby sme ho mohli prepnúť počas behu programu).
void *vlakno(struct argument_vlakna *argument){} – po spustení každú sekundu zasiela z klienta keep alive paket.
void *vlakno_vypnutie(void *vargp){} – kontroluje, či medzi klientom a serverom pri zasielaní správy nevypadne spojenie. Ak klient niečo zašle server, to skontroluje (potvrdí) a čaká na ďalšiu. Pri čakaní zapne časovač a ak 3 sekundy nič nepríde, indikuje odpojenie klienta. Tak isto to rieši s keep alive paketmi. Ak sa 3 sekundy neozve server, indikuje jeho odpojenie.
u**nsigned short getCRC(unsigned char *ukazovatel_na_data, unsigned int dlzka_spravy){} –** Slúži na výpočet CRC.
void *vlakno_KLIENT(void *vargp){}- spúšta vlákno na ktorom je celá funkcionalita klienta.
void *vlakno_SERVER(int port){} – spúšta vlákno na ktorom je celá funkcionalita servera.
int main() {} – spúšta základný program (prvotná voľba užívateľa)

Jozef Varga, ID: 86060

Používateľská príručka

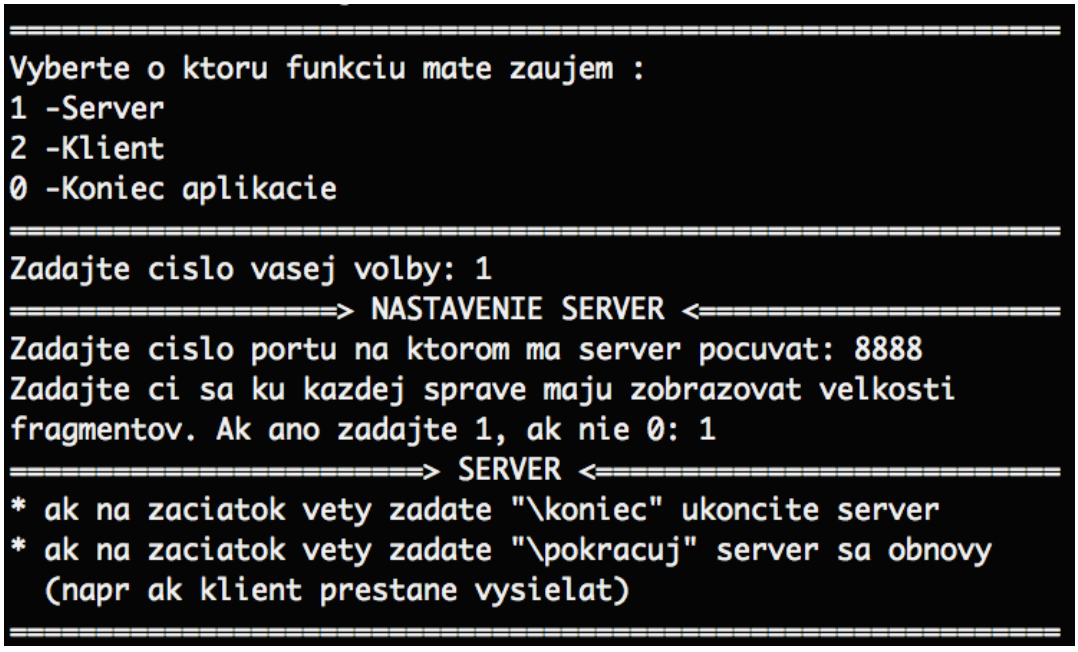
Po štarte aplikácie sa užívateľovy zobrazí nasledujúce menu:



Obrázok 10 Zapnutie aplikácie (výber funkcionality)

Užívateľ má možnosť vybrať si či aplikáciu chce spustit ako Klient - 2 (zasiela správy na server), alebo ako Server - 1(príma správy od Klienta). Treťou možnosťou je ukončiť aplikáciu - 0.

Ak aplikáciu užívateľ spustí ako Server (1) dostane možnosť nastaviť port na ktorom server čaká na správu a či chcete aby zobrazovalo aj veľkosť prenesených fragmentov správy . Ak vami nastavený port už je obsadený, aplikácia oznámy chybu a ukončí sa.



Obrázok 11 Nastavenie Servera

Po nastavení servera už len čakáte na prijatie správy. Ak vám príde obyčajná správa a máte zapnuté vypisovanie veľkostí fragmentov zobrazí sa vám takto:

```
========================> SERVER <============================
Fragment 1 má velkost: 18
Prisla sprava od: 127.0.0.1 : 53018
Sprava: Ahoj ako sa mas ?
------------------------------------------------------------
```

Obrázok 12 Server prijíma obyčajný text

Na danej správe je možné vidieť IP adresu klienta a jeho port z ktorého bola daná správa zaslaná.

Pri zaslaní umelej chyby (alebo ak sa vám chyba pri prenose vytvorí) vypíše sa vám text „!!!Chyba v kontrolnom sucte!!!“ ktorá signalizuje chybu. Tu si môžete všimnúť že fragment bol odoslaní opäť.

```
Fragment 1 má velkost: 21
!!!Chyba v kontrolnom sucte!!!
Fragment 1 má velkost: 21
Prisla sprava od: 127.0.0.1 : 53018
Sprava: Ideme dnes do kina ?
------------------------------------------------------------
```

Obrázok 13 Prijatie chybného rámca

Pri zaslaní súboru sa vám správa zobrazí následovne:

```
------------------------------------------------------------
Fragment 1 má velkost: 156
Prisiel subor od: 127.0.0.1 : 53018
Subor najdete pri aplikacii pod menom prijate
------------------------------------------------------------
```

Obrázok 14 Prijatie súboru

Súbor bude uložený pri aplikácii pod menom prijate.

Ak aplikáciu užívateľ spustí ako Klient (2), musí zadať IP adresu servera v tvare x.y.z.w (x,y,z,w sú čísla v desiatkovej sústave), číslo portu na akom server počúva, veľkosť fragmentov správy a ako posledné si užívateľ vybera či chce zobrazovať veľkosti fragmentov prenesenej správy. Pri výbere Klienta sa vypíše aj menu ktoré informuje o možnosti zasielaných správ

Po pripojení klienta sa na servery zobrazí táto správa:

```
============================================================
Klient bol pripojeny!!!
------------------------------------------------------------
```

Obrázok 15 Pripojenie klienta

Po jeho odpojení zase takáto správa:

```
------------------------------------------------------------
Klient bol odpojeny!!!
------------------------------------------------------------
```

Obrázok 16 Odpojenie klienta

```
====================> NASTAVENIE KLIENT <====================
Zadajte IP adresu servera: 127.0.0.1
Zadajte cislo portu servera: 8888
Zadajte velkost fragmentu spravy: 1000
Zadajte ci sa ku kazdej sprave maju zobrazovat velkosti
fragmentov. Ak ano zadajte 1, ak nie 0: 1
=======================> KLIENT <=========================
* ak na zaciatok vety zadate "\chyba" zaslete chybny ramec
* ak na zaciatok vety zadate "\koniec" ukoncite program
* ak na zaciatok vety zadate "\subor" dostanete sa do menu
  zasielania suboru
============================================================
```

Obrázok 17 Nastavenie Klienta

V menu je vysvetlené ako sa posiela chybný rámec alebo súbor.
Pri zadaní čistého textu bez pridania špecialnych znakov z menu klient zašle obyčajnú textovú správu:

```
============================================================
Zadaj spravu : Ahoj ako sa mas?
Fragment 1 má velkost: 17
------------------------------------------------------------
```

Obrázok 18 Odoslanie obyčajnej správy

Ak klient pred tým napíše „\chyba“ dostane sa do odosielania chybného rámca:

```
------------------------------------------------------------
Zadaj spravu : \chyba
Zadaj spravu (chybny ramec): Ideme dnes do kina ?
Fragment 1 má velkost: 21
Fragment 1 má velkost: 21
------------------------------------------------------------
```

Obrázok 19 Odosielanie chybného rámca

Ak napíše „\subor“ dostane sa do menu odosielania súboru. V tomto menu musí zadať užívateľ celú cestu (aj s koncovkou súboru) a súbor sa odošle.

```
------------------------------------------------------------
Zadaj spravu : \subor
Zadajte prosim celu cestu k suboru: /Users/jofy/Desktop/neposlem.txt
Fragment 1 má velkost: 156
------------------------------------------------------------
```

Obrázok 20 Odosielanie súboru

Ak klient zašle správu a server neodpovie vypíše hlášku „Server nie je aktívny“.

Jozef Varga, ID: 86060

Zhodnotenie

Program umožňuje jednosmernú komunikáciu z klienta na server. Taktiež dáva možnosť nastaviť klientovi IP adresu a port servera. Program umožňuje zmenu veľkosti fragmentu, pri odosielaní správ umožňuje zaslanie paketu s umelou chybou. Je tu možnosť zaslania súboru aj indikáciu ukončujúceho spojenia (odpojený klient/ server). Myslím si, že tento program plní všetky podmienky zadania a mnou navrhnutý protokol je dostatočne efektívny a spoľahlivý pre tento typ správ. Toto zadanie ma naučilo lepšie pracovať so socketmi v jazyku C a ukázalo mi mnohé problémy, ktoré sa pri prenose údajov môžu vyskytnúť a taktiež ich nasledovné riešenie.

Jozef Varga, ID: 86060

Bibliografia

Kurz CCNA zo stránky netacad.com
Materiály k zadaniu z dokumentového servera (AIS)
Prednášky z PKS – doc. Ing. Ivan Kotuliak, PhD.
Kniznica sys/socket.h http://pubs.opengroup.org/onlinepubs/7908799/xns/syssocket.h.html