**Komunikácia s využitím UDP protokolu**

Peter Šípoš

Počítačové a komunikačné siete

Cvičenie: štvrtok 10:00

**Zadanie úlohy**

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 takú, aby neboli znova fragmentované na linkovej vrstve.

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 chybných rámcov, vrátane pozitívneho aj negatívneho potvrdenia. Pri nečinnosti komunikátor automaticky odošle paket pre udržanie spojenia každých 60-120s. Odporúčame riešiť cez vlastne definované signalizačné správy.

**Program musí mať nasledovné vlastnosti (minimálne):**

1. Program musí byť implementovaný v jazyku C/C++ s využitím knižníc na prácu s UDP socket, skompilovateľný a spustiteľný v učebniach. Odporúčame použiť knižnicu *sys/socket.h* pre linux/BSD a *winsock2.h* pre Windows. Použité knižnice a funkcie 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 správy musí používateľovi umožniť určiť cieľovú IP a port.

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

* 1. 5. Obe komunikujúce strany musia byť schopné zobrazovať:
     1. a. poslanú resp. prijatú správu,

b. veľkosť fragmentov správy.

* 1. 6. Možnosť odoslať minimálne 1 chybný fragment (do fragmentu je cielene vnesená chyba, to znamená, že prijímajúca strana deteguje chybu pri prenose).
  2. 7. Možnosť odoslať súbor a v tom prípade ich uložiť na prijímacej strane ako rovnaký súbor. Akceptuje sa iba ak program prenesie 1MB súbor do 30s bez chýb.

**Analýza úlohy**

Zadanie síce hovorí, aby sme zostrojili komunikátor založený na protokole UDP, no ďalej vyžaduje viaceré funkcionality typické pre protokol TCP. Tieto funkcionality bude teda nutné odsimulovať, respektíve vlastnoručne implementovať. Medzi požadované funkcionality patria:

* Usporiadanie správ podľa poradia – ide o typickú funkcionalitu protokolu TCP. Vyslané správy môžu ísť po sieti rôznymi cestami a teda sa môžu k prijímateľovi dostať v rozličnom poradí. Aby nedošlo k problémom, u prijímateľa bude nutné zabezpečiť zoradenie prijatých správ do správneho poradia.
* Kontrola chýb pri prenose – ide o ďalšiu typickú funkcionalitu protokolu TCP. Keďže cesta dát po sieti môže byť všelijaká a funkčnosť routerov nie vždy 100%, môže sa stať, že odoslané dáta môžu prísť s nejakou chybou, prípadne nemusia prísť vôbec. Z toho dôvodu treba implementovať kontrolu takýchto chýb.
* Znovuvyžiadanie chybných rámcov – ak by došlo k chybe pri prenose ako spomína bod vyššie, je žiaduce si chybné rámce znova vypýtať. Rovnako ide o bežnú funkcionalitu TCP.
* Potvrdenie o prijatí – aby používateľ vedel, či danú správu prijal alebo nie, je potrebné implementovať potvrdenie prijatia, či už pozitívneho ale aj negatívneho (teda neprijatia)
* Udržiavanie spojenia – po vzore TCP, je nutné najprv nadviazať spojenie medzi „klientom“ a „serverom“ a toto spojenie následne udržovať. Pri UDP totiž k spojeniu ako takému nedochádza.
* Užívateľ musí zadať požadovanú cieľovú IP adresu, port a maximálnu veľkosť odosielaných správ (fragmentov).
* Keďže sa ráta s tým, že správy sa budú fragmentovať, je potrebné vymyslieť, ako bude samotná fragmentácia fungovať a ako sa budú následne fragmenty zase spájať do pôvodnej správy.
* Vzorová ukážka odoslania a prijatia chybného rámca
* Odosielanie súborov – používateľ musí mať možnosť odosielať celé súbory (aj cez fragmenty). Minimálna požiadavka je prenos 1MB súboru do 30s bez chýb.

**Návrh riešenia**

**Hlavičky**

Aby program vedel, aké dáta práve prijal a aby vedel, ktorú obslužnú funkciu zavolať a ako s dátami naložiť, bude UDP datagrama na začiatku jeho payloadu jedna z mojich vlastných hlavičiek popísaných nižšie.

Hlavička textu/súboru

Ako prvé bude pole TYPE. Zaberať bude 1B a môže nadobúdať hodnoty 1,2,3,4,5. Hodnota 1 indikuje, že datagram prenáša iba obyčajnú textovú správu (prípadne jej časť). Hodnota 2 hovorí, že prenášaný je súbor (prípadne jeho časť) a hodnota 3, že prenášaný je iba názov daného súboru. 4ka znamená, že ide o KEEP-ALIVE správu. 5ka zase, že ide o potvrdenie ne/prijatia správy.

Nasledovať bude FRAG\_AMOUNT o veľkosti 4B. Môže nadobúdať všetky hodnoty z rozsahu 1 až UNSIGNED\_INT\_MAX. Predstavuje celkový počet fragmentov danej správy či súboru. Veľkosť 4B je určená podmienkami zadania, že je nutné mať možnosť preniesť súbor o veľkosti 1MB, pričom minimálna veľkosť fragmentu môže byť 1B. Na základe danej veľkosti bude príjemca alokovať pole pre daný počet reťazcov (veľkosť == počet reťazcov).

Ďalším polom bude FRAG\_NUMBER taktiež o veľkosti 4B. Môže nadobúdať presne tie isté hodnoty, ako pole FRAG\_AMOUNT (1 až UNSIGNED\_INT\_MAX). Vyjadruje číslo fragmentu zo série (o koľký fragment sa jedná). Dôležité pre správne usporiadanie jednotlivých fragmentov a ich opätovné znovuzloženie do pôvodnej správy/súboru. Inak povedané, jedná sa o index v hlavnom poli reťazcov u prijímateľa.

Predposledným poľom bude FRAG\_SIZE o veľkosti 2B. Bude nadobúdať hodnoty od 1 po maximálnu veľkosť fragmentu (popísaná nižšie). Vyjadruje veľkosť fragmentu odosielanej správy/súboru v B. Potrebné, aby príjemca vedel nastaviť príslušnú veľkosť buffera na prijatie a následné uloženie správy. Inak povedané, ide o veľkosť samotného reťazca, ktorý sa následne uloží do poľa reťazcov.

Posledným poľom je CHECKSUM s veľkosťou 4B. Môže nadobúdať všetky hodnoty v rozsahu 0 až UNSIGNED\_INT\_MAX. Ide o kontrolný súčet, vďaka ktorému sa zistí, či dáta ktoré prišli sú v poriadku alebo sú chybné.

Hlavička názvu súboru

Jej cieľom je oddeliť názov súboru od samotného obsahu súboru. Týmto spôsobom nemusíme riešiť, kde v posielaných dátach končí názov a začína samotný obsah, keďže u prijímateľa budú uložené v samostatných bufferoch.

Obsah je totožný s hlavičkou textu/súboru až na jeden rozdiel. Veľkosť FRAG\_AMOUNT a FRAG\_NUMBER tu budú o veľkosti 2B. Keďže sa posiela iba názov súboru, tak jeho dĺžku obmedzíme na rozumnú veľkosť. Predpokladajme teda, že žiaden súbor nebude mať názov väčší ako 2B, teda 65 025 znakov. Používateľ bude o tejto skutočnosti upovedomený správou.

Hlavička KEEP-ALIVE

Cieľom KEEP-ALIVE správy je zistiť, či je druhá strana ešte stále „na príjme“ – teda či nedošlo napríklad k nejakej chybe v spojení.

Jej prvým polom je TYPE rovnako ako pri hlavičke textu/súboru.

Druhým, a zároveň posledným, polom je OPERATION s veľkosťou 1B. Môže nadobúdať hodnoty 0 a 1. 0 predstavuje „request“, teda ide o opýtanie sa druhej strany, či je ešte v spojení. 1 zasa predstavuje „reply“, teda odpoveď typu „áno, ešte tu som“.

Hlavička ne/prijatia správy

Slúži na oznámenie odosielateľovi, že jeho správa či súbor boli alebo neboli prijaté úspešne.

Prvým polom je opäť TYPE rovnako ako pri hlavičke textu/súboru.

Ďalším polom je CONFIRMATION o veľkosti 1B. Nadobúda hodnoty 0 ,1 a 3, kde 0 predstavuje neúspešné prijatie, 1 prijatie úspešné a 2 úspešné prijatie a poskladanie kompletnej správy.

Tretím polom je FRAG\_NUMBER s rovnakým obsahom ako pri hlavičke textu/súboru (obsahuje vždy číslo posledného prijatého fragmentu). Je tu na to, aby ak bola správa prijatá neúspešne (pri prenose nastala chyba), tak aby odosielateľ vedel, ktorý fragment má znovu odoslať.

**Fragmentácia a opätovné zloženie fragmentov do pôvodného stavu**

Vytvorenie fragmentov

V prípade odosielania správy sa tá najprv načíta zo štandardného vstupu do buffera *message*, kt. má veľkosť UNSIGNED\_INT\_MAX. Následne sa alokuje pole reťazcov *string\_arr*, ktoré bude držať *frag\_amount* reťazcov veľkosti *frag\_size*, kde *frag\_size* je maximálna veľkosť fragmentu zadaná používatelom a *frag\_amount* je počet fragmentov, na ktorý sa spáva rozdelí (veľkosť správy/ *frag\_size* zaokrúhlené nahor). Premenná *frag\_number* bude predstavovať index v poli reťazcov (iniciálizovaná na 0).

Nasledovať bude *while* cyklus, ktorý pôjde pokial premenná *position* bude menšia alebo rovná ako dĺžka zadaného textu (teda *strlen(message)*). Vrámci neho sa najprv do *string\_arr* na pozíciu *frag\_number* najprv pomocou funkcie *strncpy* prekopíruje toľko z pôvodnej správy (*message*), koľko sa zmestí (*frag\_size*). Následne sa navýši hodnota *position* o *frag\_size* a *frag\_number* o jedna. Tým pádom sa v ďalšej iterácii uloží zase ďalšia časť správy do ďalšieho fragmentu. Slučka končí, keď *position* bude väčší ako dĺžka pôvodnej správy a teda v *string\_*arr bude uložený celý jej obsah, vrátane 0-terminátoru.

Na záver sa postupne odošle celá správa, postupným odosielaním jednotlivých reťazcov z pola *string\_arr*.

Ukážka kódu:

int frag\_number = 0;

char \*\*string\_arr;

string\_arr = (char\*\*)malloc(frag\_amount\*sizeof(char\*));

for(int i = 0; i<frag\_amount; i++)

string\_arr[i] = (char\*)malloc(frag\_size);

int position = 0;

while(position<=strlen(message)){

strncpy(string\_arr[frag\_number], message + position, frag\_size);

position += frag\_size;

frag\_number++;

}

sendto.... // odosli spravu

V prípade odosielanie súboru je postup de facto rovnaký. Akurát sa to celé odohráva ešte v jednom while cykle, ktorý beží až kým sa pri čítaní zo súboru program nedostane na jeho koniec. Zakaždým sa z neho načíta jeden riadok, ten sa uloží do buffera *message* a ďalej sa postupuje rovnako ako vyššie.

Skladanie fragmentov

Z hlavičky prvého prijatého fragmentu sa najprv prečítajú údaje v poliach FRAG\_AMOUNT a FRAG\_SIZE a uložia sa do premennej *frag\_amount,* respektíve *frag\_size*. Následne sa alokuje pole reťazcov *string\_arr*, ktoré bude držať *frag\_amount* reťazcov veľkosti *frag\_size*. Z poľa FRAG\_NUMBER sa jeho hodnota analogicky uloží do premennej *frag\_number*. Do bufferu *msg\_fragment* s veľkosťou *frag\_size* sa uloží obsah prenášanej správy (payload,časť prenášanej správy). Následne sa vypočíta kontrolný súčet z dát v *msg\_fragment* pomocou funkcie *CRC32* a jeho výsledok sa uloží do premennej *checksum\_received*. Z hlavičky sa prečíta údaj z poľa CHECKSUM a uloží do premennej *checksum\_original*. Ak sa *checksum\_received* a *checksum\_original* budú rovnať (teda prijatá správa je bezchybná), tak sa obsah prijatej správy (*msg\_fragment*) prekopíruje do *string\_arr* na pozíciu *frag\_number*. V opačnom prípade sa bude daný fragment vyžiadaný znova.

Premenná *fragments\_received* bude slúžiť ako počítadlo prijatých fragmentov. Keď príde posledný fragment, teda *fragments\_received* sa bude rovnať *frag\_amount*, tak sa kompletná správa vypíše (bude sa postupne čítať a apendovať zo *string\_arr*).

Ukážka kódu:

// najprv prijmi prvy a zneho nacitaj dolezite udaje a sprav alokacie

rcvfrom.... // prijmi spravu

int frag\_amount = FRAG\_AMOUNT;

int frag\_size = FRAG\_SIZE;

int frag\_number = FRAG\_NUMBER;

int checksum\_original = CHECKSUM;

int checksum\_received, fragments\_received;

fragments\_received = 1;

char \*msg\_fragment, \*\*string\_arr;

msg\_fragment = (char\*)malloc(frag\_size);

string\_arr = (char\*\*)malloc(frag\_amount\*sizeof(char\*));

for(int i = 0; i<frag\_amount; i++)

string\_arr[i] = (char\*)malloc(frag\_size);

strcpy(msg\_fragment, PAYLOAD);

checksum\_received = CRC32(msg\_fragment);

if (checksum\_received == checksum\_original)

strcpy(string\_arr[frag\_number], msg\_fragment);

else

request\_fragment(frag\_number);

fragments\_received++;

// potom ich prijmaj a ukladaj az kym nebudu vsetky

while(fragments\_received != frag\_amount){

rcvfrom....

frag\_number = FRAG\_NUMBER;

checksum\_original = CHECKSUM;

strcpy(msg\_fragment, PAYLOAD);

checksum\_received = CRC32(msg\_fragment);

if (checksum\_received == checksum\_original)

strcpy(string\_arr[frag\_number], msg\_fragment);

else

request\_fragment(frag\_number);

fragments\_received++;

}

printf(string\_arr);

V prípade že sa prijímal súbor, tak je postup totožný, akurát na záver sa vytvorí nový súbor a obsah *string\_arr* sa postupne zapíše doňho.

**Iné**

Nasledovať budú ďalšie poznámky, ktoré by v návrhu určite nemali chýbať, no nie sú hodné samostatnej väčšej kapitoly ako časti pred.

Veľkosť fragmentu

Maximálna veľkosť jedného fragmentu bude obmedzená na 1457 B, aby sa nepresiahlo MTU 1500B a teda aby sa zabránilo opätovnej fragmentácii. 20 B totiž zaberie IP hlavička, hlavička UDP 8B a moja najdlhšia hlavička 15B. Minimálna veľkosť bude 1B. Ak by bola zadaná veľkosť fragmentu z mimo požadovaného rozsahu (teda menšia ako min alebo väčšia ako max), program vyhlási chybu a požiada používateľa o opätovné zadanie veľkosti.

Checksum a jeho výpočet

Na overenie správnosti dát bude využité ich overenie pomocou kontrolného súčtu. Ten bude realizovaný známou funkciou CRC32. Jej vstupom sú nejaké dáta (správa) a výstupom je unsigned\_32bit číslo.

Funkcia funguje na princípe binárneho delenia určitým polynómom. Vstup sa dorovná počtom núl rovnému stupňu daného polynomiálu (v tomto prípade 32-1). Následuje bitové delenie v modulo 2. Výsledkom funkcie je zvyšok po tomto delení.

Kontrola prijatia všetkých fragmentov

Počas odosielania dát je možné, že sa občas nejaký fragment stratí, či už v dôsledku nejakého digitálneho šumu, silného elektromagnetického poľa alebo čohokoľvek iného. V konečnom dôsledku nezáleží na tom, prečo sa fragment stratil, ale *že* sa stratil. Pretože v takom prípade nemôže dôjsť k znovuvyskladaniu správy. Preto je nutné takýto prípad ošetriť.

Po každom prijatom fragmente príjemca kontroluje, či už je pole reťazcov, kam fragmenty ukladá, plné. Pole je plné, keď sa premenná *fragments\_received* rovná *frag\_amount.* Ak áno, tak fragmenty prišli všetky a program pokračuje štandardne ďalej.

Ak do 5 sekúnd nepríde žiadny fragment, príjemca prehladá pole reťazcov kým nenájde prvý volný index a pošle potvrdenie neprijatia s CONFIRMATION 0 a FRAG\_NUMBER rovným prvému volnému indexu v poli. Je totiž zrejmé, že fragment s daným číslom chýba, a teda prijímateľ si ho opätovne vyžiada.

Potvrdenie ne/prijatia

Po odoslaní každého fragmentu príjemca očakáva potvrdenie ne/prijatia od príjimateľa. To je realizované prostredníctvom odoslanie jednoduchého packetu, ktorý obsahuje moju hlavičku ne/prijatia správy. Ak bola správa prijatá úspešne, príjmateľ odošle správu úspešného prijatia (hodnota CONFIRMATION v hlavičke bude 1). V opačnom prípade, teda ak nesedeli vypočítaný a prijatý kontrolný súčet, bude hodnota CONFIRMATION 0 a následne bude opätovne vyžiadaný príslušný fragment.

V prípade úspešného prijatia fragmentu o tom používateľ nebude informovaný v snahe predísť zahlteniu konzoly používateľa (napr pri posielaní 1MB súboru po 1B) a rovno sa pošle ďalší fragment.

Ak sa príjemcovi podarilo prijať a vypísať kompletnú správu, pošle odosielateľovi správu o prijatí s hodnotou CONFIRMATION == 2. Ten následne vypíše, že daná správa bola úspešne prijatá.

Keep-alive, respektíve udržanie spojenia

Počas komunikácie sa môže stať všeličo. Odosielateľ alebo príjmateľ sa môžu hocikedy z neznámych príčin odpojiť alebo dokonca aj vypnúť, môže byť prerušené spojenie, ba môže padnúť aj celá sieť. Aby program zbytočne nebežal do nekonečna a nebral tak zbytočne prostriedky počítaču, zavedieme funkcionalitu keep-alive. V zásade ide o jednoduché opýtanie sa druhej strany, či je ešte tam.

Ak príjimateľ po dobu 20 sekúnd nepríjme žiadnu správu, pošle odosielateľovi keep-alive request a na konzolu vypíše správu „Haló, je tam niekto?“. Ak odosielateľ príjme tento request, na konzolu vypíše rovnakú správu ako vypísal príjmateľ, potom ďalšiu správu v tvare „Áno, som tu, len mi to trochu trvá.“ a odošle príjmateľovi keep-alive reply. Keď ho príjmateľ príjme, tak vypíše odpoveď odosielateľa („Áno, som tu...“) a ďalej čaká na príjem dát.

Podobne, ak odosielatel po dobu 20 sekúnd nepríjme žiadne potvrdenie prijatia fragmentu, odošle keep-alive request príjmateľovi a scenát je rovnaký ako je popísané vyššie, len sa vymenia role príjmateľa a odosielateľa.

V oboch prípadoch, ak do 10 od prvého keep-alive requestu nepríde odpoveď, pošle sa druhý. Ak nepríde odpoveď do 20 sekúnd, pošle sa tretí. Ak nepríde odpoveď do 30 sekúnd, program prehlási spojenie za nekorektne ukončené a skončí. Samozrejme o tom informuje používatela správou typu: „Druhá strana je neresponzívna. Zrejme došlo k chybe v spojení. Program sa ukončuje. Za vzniknuté problémy sa ospravedlňujeme.“

Skúška a priebeh zachytenia chybného fragmentu

Na overenie správnosti fungovania detekcie chybných fragmentov bude mať používateľ možnosť schválne odoslať chybný fragment. Po kompletnom načítaní programu vo funkcii odosielateľa (po zadaní cieľovej IP adresy a iných) bude používateľ s touto možnosťou konfrontovaný. Po jej zvolení bude vyzvaný, aby zadal nejaký text. Program následne pri ukladaní fragmentov do hlavného poľa reťazcov do náhodného fragmentu uloží miesto kontrolného súčtu samé nuly a správu pošle ako normálne.

Príjemca správu prijíma ako normálne. No keď narazí na onen schválne zlý fragment, tak zistí, že checksum v príslušnej hlavičke sa nezhoduje s checksumou vypočítanou dát. V tom prípade pošle odosielateľovi potvrdenie o neprijatí správy, v ktorého hlavičke je číslo poškodeného fragmentu.

Príjemca po každom odoslanom fragmente očakáva potvrdenie od príjemcu, že fragment dorazil správne. Ak sa tak stane, tak pošle ďalší fragment. No v tomto prípade vypíše správu „Fragment s číslom XY nebol prijatý úspešne. Posielam ho teda znovu.“ A znovu odošle fragment s príslušným číslom fragmentu. Teda *frag\_number* sa nastaví na to z hlavičky, a načíta sa a odošle príslušný prvok z hlavného poľa reťazcov.

Stručný priebeh fungovania programu

Po spustení programu používateľa privíta otázka „Akým spôsobom chcete spustiť program? Ako prijímateľa, alebo odosielateľa? Pre prijímateľa zadajte „p“, pre odosielateľa zasa „o“. Ak si prajete program ukončiť, zadajte „k“. “

Po vybratí si režimu odosielateľa bude nasledovať požiadavka, aby používateľ zadal cieľovú IP adresu, port a maximálnu veľkosť fragmentov. V správe bude používateľ upozornený na minimálnu a maximálnu veľkosť fragmentu (1B a 1457).

Na záver bude používateľ vyzvaný, aby sa rozhodol, či chce odoslať iba obyčajnú správu, chybnú správu alebo textový súbor.

Ak bude chcieť odoslať iba textovú správu, bude vyzvaný aby zadal „text“, následne napísal samotnú správu a enterom ju odoslal. Rovnako sa postupuje aj v prípade, ak chce používateľ odoslať správu so schválne vloženou chybou.

Ak bude používateľ chcieť odoslať súbor, bude vyzvaný, aby zadal cestu k danému súboru a zároveň bude upozornený, že odosielať sa dajú iba textové súbory.

V každom prípade bude odosielanie končiť (teda po odoslaní posledného fragmentu) správou od programu, že správa sa úspešne odoslala a že sa čaká na potvrdenie prijatia od prijímateľa.

V prípade, že užívateľ zvolil režim prijímateľa, bude vyzvaný aby zadal číslo portu. Následne sa už len vypíše správa, že program čaká na správu od odosielateľa. Po úspešnom prijatí kompletnej správy túto správu vypíše. V prípade prijatia textového súboru používateľovi oznámi, že sa úspešne prijal súbor s názvom ABCD a že je dostupný v priečinku, kde je uložený samotný program (kvôli zjednodušeniu implementácie).

Na úplný záver, teda po vypísaní správy potvrdzujúcej úspešné prijatie u odosielateľa a vypísaní prijatej správy, respektíve správy o prijatom súbore, bude používateľ konfrontovaný s ďalšou voľbou. „Prajete si, aby program zotrval v aktuálnom režime? Ak áno, zadajte „ano“, v opačnej prípade zadajte „nie“.“ Ak používatel zadá áno, tak program pokračuje ako pred tým (v prípade príjmateľa vypíše že čaká na správu, v prípade odosielateľa opäť ponúkne 3 možnosti odoslania). Ak používateľ zadá nie, program sa vracia na úvodnú voľbu „Akým spôsobom chcete spustiť program? Ako prijímateľa, alebo odosielateľa? Pre prijímateľa zadajte „p“, pre odosielateľa zasa „o“.“ a ďalej pokračuje ako odosielateľ/príjemca.

Z programátorského hladiska sa budú výbery jednotlivých možností realizovať pomocou podmienok *if* a následným volaním príslušných funkcií. Samozrejmosťou je ošetrenie chybných vstupov. Napr:

while (1)

scanf(“%c”, &mode);

if (mode == ‘o’)

sender();

continue;

else if (mode == ‘p’)

receiver();

continue;

else if (mode == ‘k’)

break;

else

printf(“Zadali ste nevalidny mod. Zadajte prosim “o” pre spustenie odosielatela, “p” pre prijmatela.\n”);

alebo

scanf(“%s”, operation);

while(1)

if(strcmp(operation, “text”) == 0)

send\_text();

break;

else if (strcmp(operation, “chybny”) == 0)

send\_corrupted();

break;

else if (strcmp(operation, “subor”) == 0)

send\_file();

break;

else

printf(“Zadali ste nevalidnu operaciu. Pre odoslanie textovej spravy zadajte “text”, pre otestovanie zachytenia chybneho fragmentu zadajte “chybny” a pre odoslanie textoveho suboru zadajte “subor”.\n”)

scanf(“%s”, operation);

**Zmeny v implementácii oproti návrhu**

Hlavičky

V programe je síce implementovaná hlavička pre keep-alive, no samotná funkcia už nie, z dôvodu nedostatku času.

Hlavička názvu súboru bola kompletne zrušená. Ukázalo sa totiž, že nie je reálne potrebná. Reálna pamäťová úspora by bola 4B, čo by sa jednoznačne neoplatilo vzhľadom k dĺžke implementácie. Namiesto toho bola funkcia prijímania rozlíšená na prijímanie textu a príjem súboru.

Pole Confirmation v hlavičke ne/prijatia správy nadobúda iba hodnoty 0 a 1. Úspešné prijatie totiž indikuje aj úspešné poskladanie kompletnej správy, keďže hodnota 1 sa odošle iba v prípade, že už úspešne prišli všetky fragmenty, a správa či súbor boli zrekonštruované. Ušetrujeme tak linku od zbytočného posielania packetov.

Fragmentácia

Dĺžka jednotlivého alokovaného reťazca v hlavnom bufferi je *frag\_size + 1*. Treba totiž vyhradiť aj priestor na terminátor reťazca.

Podmienka vo while slučke bola nahradená logickejším výrazom position < strlen(message) + 1 . Potrebujeme totiž iterovať, kým pozícia v zadanej správe bude menšia ako samotná dĺžka správy, no musíme pridať aj 1, keďže potrebujeme poslať aj terminátor reťazca.

Bol pridaný krok, kedy sa na posledný index vo fragmente vloží terminátor reťazca, ktorý by tam inak nebol, keďže *strncpy* ho tam sám nevloží.

Fragmentácia súboru sa líši v tom, že while cyklus v ktorom sa to odohráva nebeží kým sa pri jeho čítaní nedostaneme na jeho koniec, ale v nekonečnom cykle. Zo súboru nečítame po riadkoch, ale po blokoch o veľkosti veľkosti fragmentu zadanej používateľom pomocou funkcie *fread*. V pôvodnom prípade by totiž fragmenty nemuseli mať požadovanú dĺžku. Slučka končí, keď *fread* vráti nulu, a teda sa dočítalo až po koniec súboru.

Skladanie fragmentov

Dĺžka jednotlivého alokovaného reťazca v hlavnom buffri je *frag\_size + 1*. Treba totiž vyhradiť aj priestor na terminátor reťazca.

*Fragments received* začína na 0, keďže na po prijaté prvého ich zväčším o 1.

*Msg\_fragment* zanechaný – nepotrebný. Slúžil na kopírovanie prijatej správy do hlavného buffera. To sa ale dá rovno z prijatých dát a netreba na to separé string.

Upravená podmienka na základe ktorej sa správa zapíše do hlavného buffera – ak sa checksumy nerovnajú, je navýšený počet prijatých fragmentov (fragment sa totiž naozaj prijal, len nebol správny) a pokračuje sa v príjme. Inak je obsah fragmentu zapísaný, počet prijatých fragmentov navýšený a pokračuje sa. Tento spôsob zabraňuje nekonečnému čakaniu z dôvodu, že by sa počet prijatých nerovnal počtu celkovému.

Namiesto čisto schématického printf(string\_arr) je implementovaný reálny postupný výpis bufferu vo while cykle.

Spustenie, respektíve úvod programu

V úvodnej voľbe je po vykonaní funckie *sender* a *receiver* pridaný *getchar* na to, aby „zhltol“ newline z potvrdenia voľby používateľom.

Po potvrdení, že používateľ chce pokračovať v danom režime v prípade príjemcu program nepokračuje automaticky v prijímaní na tom istom porte a tej istej veci (textu/súboru), ale dá používateľovi opäť do ponuky, že čo chce prijímať a na akom porte. Jednak je program takto trochu viac „znovapoužitelný“, no hlavne, na presnú implementáciu podľa návrhu nezostal čas.

Kontrola prijatia všetkých fragmentov

Pôvodný plán by bol veľmi neefektívny pri posielaní dlhých správ, respektíve súborov, keďže neustála kontrola hlavného buffera a žiadosť o znovu poslanie fragmentu by bola príliš častá.

Konečná implementácia rieši kontrolu prijatia za pomoci funkcie *select* v role timeoutu. Ak sa totiž stane, že nejaký fragment sa prosto stratí a nikdy nedorazí do cieľa, tak hlavná slučka by nikdy neskončila, keďže príjemca by stále čakal na ďalšej fragmenty, no odosielateľ ich už mal všetky odoslané. Keďže funkcia na prijímanie *recvfrom* je blokujúca funkcia, bez nejakého timoutu by sa z nej nebolo možné dostať. Preto je implementovaný.

Časy čakania sú tiež rôzne. Čakať 5s na každý možný stratený fragment by bolo pridlho. Čaká sa teda 10s na prvý fragment (aby mal používatel čas zadať údaje odosielateľovi) a na všetky ostatné len 0,1ms, aby bol príjem naozaj čo najplynulejší. Vždy, ak časovač vyprší (a teda žiadny fragment nedorazil v danom časovom okne), bude počet fragmentov aj tak navýšený, pretože rátame s tým, že nejaký mal prísť, no neprišiel. Inak by totiž hlavná slučka išla do nekonečna, keďže počet prijatých by sa nikdy nedostal na úroveň počtu všetkých, ako je popísané vyššie.

Kontrola prijatia všetkých fragmentov sa deje následne iba raz, jedným prehladaním hlavného pola reťazcov. Ak niektorý fragment chýba (na prvom mieste daného reťazca je 0), tak program tento fragment vyžiada a následne ho pridá do poľa.

Tento znovuvyžiadný fragment určite príde. O to sa stará funkcia *request\_fragment.* Tá požaduje fragment s daným číslom až kým nebol prijatý správne. Rovnako ako v prípade prijatia hocijakého fragmentu, aj tu je implementovaný timeout, keďže vyžiadaný fragment nemuší doraziť.

Zmeny v potvrdení ne/prijatia sú popísané vyššie.

Skúška a priebeh zachytenia chybného fragmentu

Chybný je vždy iba prvý fragment, keďže na implementácii pôvodného nápady (chybný náhodný fragment) nebol čas. Rovnako ako je popísané vyššie, správa o chybnom prijatí a znovyžiadanie si chybného fragmentu sa deje až po prijatí všetkých kvôli vyššie spomenutým dôvodom. Aby nebolo nutné implementovať samostatnú funkciu detegovania chyby pre skúšobný príklad, využíva sa jedna všeobecná, ktorá neimplementuje žiadne informatívne výpisy typu „fragment nebol prijatý, posielam ho znova“ z dôvodu výrazného spomalenia programu. Keď sa má totiž takáto správa vypísať niekoľko tisíc krát pri posielaní väčšieho súboru, tak to naozaj trvá.

Posielanie súboru

Pri odosielaní súboru sa nezadáva celá cesta k súboru, ale len jeho názov. Súbor pritom musí byť umiestnený tam, kde aj aplikácia. Je to tak preto, aby sa mohol jednoducho odoslať iba názov súboru. V pôvodnom prípade by sa najprv z celej cesty musel vystrihnúť iba samotný názov súboru, na implementáciu čoho nebol čas.

Maximálna dĺžka názvu súboru je obmedzená na maximálnu veľkosť fragmentu a posiela sa iba v jednom packete (nefragmentuje sa), aby sa následne nemusel v príjemcovi súboru implementovať de facto rovnaký kód ako v príjemcovi textu, keďže najprv by som musel prijať všetky fragmenty textu, potom dať ten názov znova dokopy a až potom by sa mohol daný súbor vytvoriť.

Iné

Veľkosť textovej správy (a teda veľkosť buffera do ktorej sa načíta) je implementačne obmedzená na 10 000 znakov. Nepredpokladá sa, že by používatel chcel zadať dlhšiu správu. Len pre predstavu, bola by to skoro polovica tohto dokumentu natlačená v jednom riadku. Ak by sa mal totiž buffer alokovať dynamicky iba na velkosť správy, bolo by to implementačne zložité a zadanie by sa nestihlo načas odovzdať.

Veci, ktoré neboli v návrhu spomenuté ale sú implementované (okrem už vyššie spomenutých)

Program poctivo zachytáva možné errory a dáva užívateľovi správu, o aký error sa jedna.

Vytváranie socketu, napĺňanie adresy, bindovanie

Po úspešnom prijatí prvého fragmentu sa vypíše jeho veľkosť (teda veľkosť všetkých prichádzajúcich fragmentov) – minimálna podmienka zadanie 5.b.