

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLOGIÍ

Zadanie 2

KOMUNIKÁCIA S VYUŽITÍM UDP PROTOKOLU

Anna Yuová

Predmet: Počítačové a komunikačné siete
Akademický rok: 2020/2021
Semester: zimný

ZADANIE

Navrhните a implementujte program s použitím vlastného protokolu nad protokolom UDP (User Datagram Protocol) transportnej vrstvy sieťového modelu TCP/IP. Program umožní komunikáciu dvoch účastníkov v lokálnej sieti Ethernet, teda prenos textových správ a ľubovoľného súboru medzi počítačmi (uzlami).

Program bude pozostávať z dvoch častí – vysielacej a prijímacej. Vysielací uzol pošle súbor inému uzlu v sieti. Predpokladá sa, že v sieti dochádza k stratám dát. Ak je posielaný súbor väčší, ako používateľom definovaná max. veľkosť fragmentu, vysielajúca strana rozloží súbor na menšie časti - fragmenty, ktoré pošle samostatne. Maximálnu veľkosť fragmentu musí mať používateľ možnosť nastaviť takú, aby neboli znova fragmentované na linkovej vrstve. Ak je súbor poslaný ako postupnosť fragmentov, cieľový uzol vypíše správu o prijatí fragmentu s jeho poradím a či bol prenesený bez chýb. Po prijatí celého súboru na cieľovom uzle tento zobrazí správu o jeho prijatí a absolútnu cestu, kam bol prijatý súbor uložený.

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

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 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.
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.

OPIS RIEŠENIA

Na začiatku programu zistím, kto ide komunikovať – rozlišujem server a klient (ako prvý musí začať server). Keď už viem, kto komunikuje, musím si načítať ip adresu a porty, na ktorej idú komunikovať. Po načítaní portov a ip adries idem skontrolovať či bolo nadviazané spojenie cez „2 way handshake“. Ako prvý začne server, ktorý sa zapne a čaká pokiaľ sa nezapne aj klient. Ak sa klient nezapne do 40 sekúnd, server sa vypne (udržiavanie keep alive). Ak sa klient zapne, vytvorím si správu, ktorú odošle serveru. Ak sa tá správa zhoduje aj v servery, potvrdím klientovi, že spojenie bolo nadviazané a pokračujem v programe, ak sa správy nezhodujú, tak sa vypnú – pokiaľ to ručne nezmením, tak sa budú vždy zhodovať. Rozdiel oproti 3 way handshake je v tom, že po nadviazaní spojenia už klient neposiela znovu správu serveru, že je to v poriadku.

Po nadviazaní spojenia pokračujem v klientovi, kedy má na výber, či chce poslať správu (text zo vstupu), súbor alebo chce skončiť.

Ak chce poslať správu:

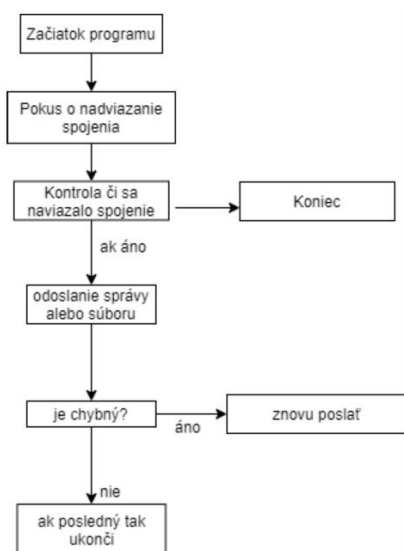
Musí si najprv načítať danú správu zo vstupu. Potom si musí načítať ešte veľkosť, podľa ktorej ju chce rozfragmentovať. Podľa dĺžky správy a počtu fragmentov si zistím, na koľkokrát sa správa bude posilať a potom ju odosiela postupne vo fore. Najskôr si vytvorím hlavičku, do ktorej nastavím TYP – rozlišujem či ide napr. o správ alebo súbor, potom VELKOSŤ je veľkosť prenesených bajtov, potom PORADIE – poradové číslo fragmentu, POČET FRAGMENTOV – koľko fragmentov dokopy sa bude posilať a CRC výpočet na základe CR-16 funkcie. Túto hlavičku si posielam spolu s dátami (hlavička + dáta) a na druhej strane si ju musím rozbaľiť (unpack) a oddelím si časť, ktorá sa týka hlavičky a časť, ktorá sa týka dát. Hlavička má veľkosť 20 – 1 char (TYP) a 4 inty(ostatné) - ciiii. Po oddelení hlavičky na druhej strane viem na základe údajov v hlavičke zistiť rôzne parametre – napr. TYP je nultý argument hlavičky (hlavička[0]), a tak si postupne uloží jednotlivé premenné. Podľa typu viem rozlíšiť či sa poslal súbor alebo správa alebo či chce skončiť. Ak sa poslala správa typ je b'p' a ak súbor typ je b's'. Ak posila správu postupne sa posielajú z klienta na server všetky fragmenty za sebou (metóda posielania všetkých naraz). Pri prijímaní na druhej strane si ich ukladám buď priamo do výsledného pola alebo si uloží indexy chybných. Ak posielam dobré, skontrolujem crc, či sa rovná crc číslo z hlavičky a novo vypočítané u servera a rovno to uloží do výsledného pola, ak došli chybné, u klienta si naschvál zmením crc a u servera si teda iba uloží poradové číslo tých fragmentov, ktoré boli chybné. Keď som už poslala všetky naraz, tak si skontrolujem, či došli všetky, ktoré očakával (počet fragmentov z hlavičky musí sedieť s poradím, koľko ich došlo) a ak náhodou nie sú všetky, server pošle klientovi pole s chýbajúcimi vynechanými indexmi fragmentov. Klient ich pošle znovu a server zas skontroluje či prišli správne alebo došli pokazené. Ak už má všetky, ktoré očakával, idem si vyžiadať tie, ktoré boli chybné ešte raz. Ak náhodou ani jeden fragment nebol chybný, rovno vypíše posielanú správu a ak bol nejaký chybný, vyžiada si znovu indexy tých chybných a klient mu ich znovu pošle už v dobrom stave a potom vypíše výslednú správu. Správu si postupne po fragmentoch ukladám na key (poradové číslo) a val je daný poslaný fragment. Na konci pri výpise si ich zoradím podľa keys a vypíšem zoradenú kompletnú správu.

Ak chce poslať súbor:

Pri posielaní súboru si najprv načítam názov súboru a veľkosť fragmentov zo vstupu. Najskôr si na druhú stranu pošlem nový názov súboru, ktorý sa vytvorí po odoslaní (napr. ak načítam menoSuboru.txt tak na druhú stranu sa vždy pošle menoSuboru2.txt a to bude tá moja poslaná správa). Potom pokračujem rovnako ako pri posielaní správy – vytvorím si hlavičku a posielam jednotlivé fragmenty spolu s hlavičkou, na druhej strane si ich rozbaľím a analyzujem. Ak došli správne rovno sa uložia do výsledného pola (toto pole už nie je

pomocou dict ale pomocou funkcie insert, vkladám na poradie dané dáta) alebo ak došli nesprávne, uloží si ich indexy. Takto si postupne naposielam všetky fragmenty za sebou. U súboru rátam s tým, že vždy dorazia všetky naraz a žiaden nebude vynechaný. Keď už prídu na server všetky fragmenty, idem skontrolovať, ktoré boli chybné a rovnako ako pri správe si pošlem ku klientovi pole indexov chybných správ, u klienta im pošlem crc aké majú mať a pošlem ich znovu ako dobré.

Klient:



Server:



SPUSTENIE A TESTOVANIE

Program som implementovala v jazyku Python, použila som program Pycharm (verzia 2020.2.3) a je spustiteľný.

Na začiatku programu si otvorím terminál, cez ktorý simulujem 2 počítače – local(1) a local(2). Po zadaní do local(1) alebo (2) – nezáleží na poradí zadám „main.py“ a zobrazí sa na výber, či chce byť server alebo klient. Pre správnu funkčnosť programu sa musí zapnúť server ako prvý. Do druhého local teda napíšem main.py a klient a ak to stihnem do 40 sekúnd, spojenie sa nadviazalo a pokračujem. Ešte predtým si ako si pri klientovi zadá čo chce robiť, musí zadať ip adresu a porty – treba napísať tie, ktorú sú tam napísané.

U klienta potom musím zadať, čo chce robiť, ak chce poslať správu zadá ‚p‘, ak chce poslať súbor zadá ‚s‘ a ak chce skončiť zadá ‚k‘. Ak zadá p (správu) musí napísať na vstup textovú správu, stlačiť enter a zadať veľkosť fragmentov. U klienta sa zobrazí znovu čo chce robiť ako ďalšie a u servera sa vypíše poslaná správa. Ak zadá s (súbor) musí napísať názov súboru aj s jedno príponou (napr. zajac.png alebo pes.jpg) a veľkosť fragmentov. V priečinku, kde je aj main sa potom zobrazí nový prenesený súbor (napr. zajac2.png alebo pes2.jpg). Ak zadá k, tak program skončí na oboch stranách.

SPLNENÁ FUNKCIONALITA

- možnosť načítania ip adresy a portu zo vstupu
- optimálna práca s dátami
- možnosť zvolenia max. veľkosti fragmentovania
- simulácia chýb/chyby pri prenose
- prijímacia strana vie oznámiť chybné/neprijaté fragmenty
- simulované vynechanie paketov pri posielaní správy
- výpis absolútnej cesty k súboru a veľkosti posielaných fragmentov na oboch stranách
- klient server
- keep alive (iba na začiatku pri 3 way handshake)

ZMENA OPROTI NÁVRHU

1. Zmena metódy posielania – posielam to metódou, že pošlem všetko naraz a nie po balíkoch ako som písala v návrhu.
2. Zmena hlavičky – v návrhu som načrtla hlavičku v poradí – typ, počet poradie, informácie, odosielané dáta a crc. Hlavičku som trochu pozmenila tak, že som si dala posielané dáta na koniec za crc, aby sa mi aby ľahšie zabalila hlavička + dáta na posielanie a nemusela som to zbytočne rozdeľovať.

PRÍKLADY

Posielanie správ:

Klient

Chce poslať správu, veľkosť fragmentov je 3:

```
Terminal: Local (2) × Local × +
(venv) C:\Users\Anna\PycharmProjects\pythonProject1>main.py
Zadaj server alebo klient: klient
Zadaj port: 2021
2021
Zadaj ip: 127.0.0.1
127.0.0.1
Klient zisťuje, či komunikujú...
Spojenie je nadviazane

Zadaj s (subor), p (sprava), k (koniec):
p
Chce poslať správu
Zadaj správu:
Spotreba paliva sa podľa štandardu WLTP pohybuje okolo 6,6 litra benzínu na 100 km. Prostredníctvom prepínača Drive Mode Select môže vodič zvoliť jeden zo
4 jazdných režimov (ECO, NORMAL, SPORT alebo TRAIL). Všetky štyri režimy možno použiť aj v čase, keď sa vozidlo pohybuje v čisto elektrickom režime EV. Spo
treba paliva sa podľa štandardu WLTP pohybuje okolo 6,6 litra benzínu na 100 km. Prostredníctvom prepínača Drive Mode Select môže vodič zvoliť jeden zo 4 j
azdných režimov (ECO, NORMAL, SPORT alebo TRAIL). Všetky štyri režimy možno použiť aj v čase, keď sa vozidlo pohybuje v čisto elektrickom režime EV.
Zadaj veľkosť fragmentu:
```

Ďalej vypíše veľkosti posielaných fragmentov a na konci počet prenesených fragmentov. Ak chýbali nejaké fragmenty – vráti klientovi, ktoré indexy musí poslať znovu a potom mu ešte vráti, ktoré indexy má poslať znovu lebo boli chybné.

```
Terminal: Local (2) × Local × +
Odosiela fragment o veľkosti: 3 bajtov
Odosiela fragment o veľkosti: 3 bajtov
Odosiela fragment o veľkosti: 3 bajtov
Odosiela fragment o veľkosti: 3 bajtov
Odosiela fragment o veľkosti: 3 bajtov
Odosiela fragment o veľkosti: 1 bajtov
Dokopy odoslalo správu na: 205 fragmentov

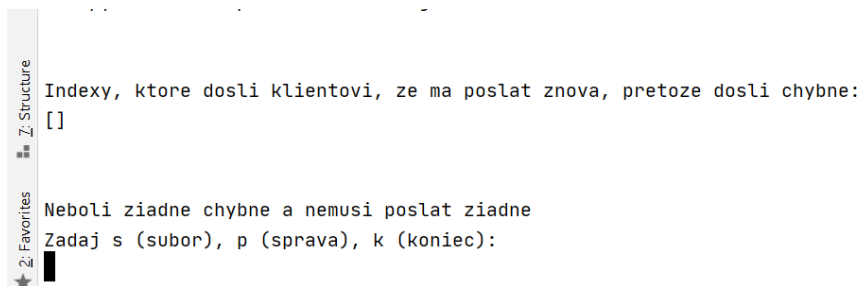
[5]
Klient posíela znovu, tie čo chýbali
Indexy, ktoré dosli klientovi, že ma poslať znova, pretože dosli chybné:
[6, 8]

Klient posíela znovu chybné
Klient posíela znovu chybné
Zadaj s (subor), p (sprava), k (koniec):
```

Server:

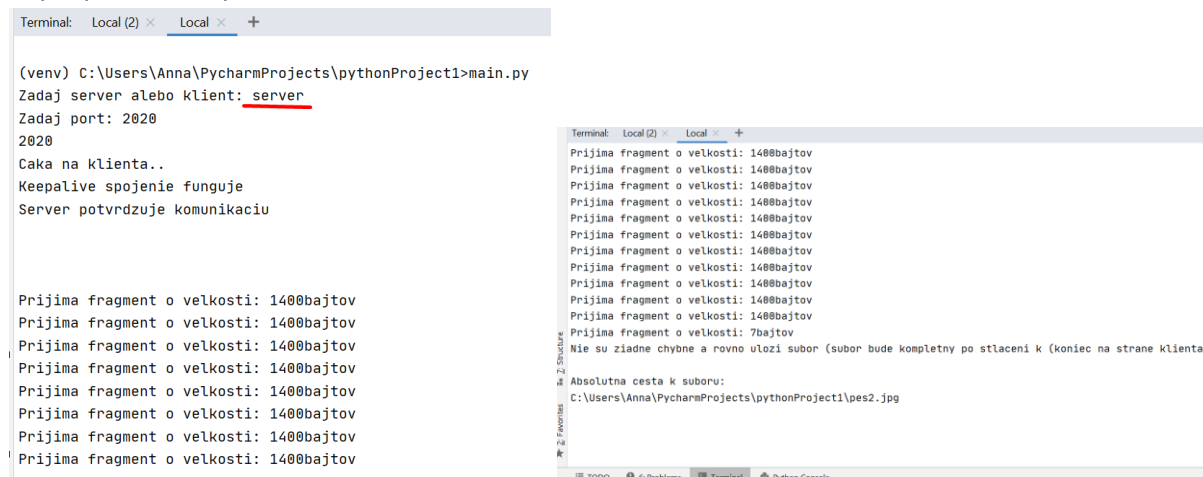
Na začiatku potvrdí, že sa spojili a skontroluje keep alive. Ďalej vypíše rovnako u klienta počet prenesených paketov a ich veľkosti.

Ďalej sú tam najprv indexy chýbajúcich, ktoré musí vyžiadať znovu a potom indexy chybných, ktoré musí tiež vyžiadať znovu. Na konci sa vypíše poslaná správa.





Server:

Vypíšem si, že bolo spojenie nadviazané a potom veľkosť prenášaných fragmentov, ak je tak pole chybných, ktoré vyžiada znovu a absolútnu cestu k súboru.



Pôvodný súbor je pes.jpg (24kb) a nový súbor – poslaný je pes2.jpg a má rovnakú veľkosť ako pôvodný.

 pes	1. 12. 2020 18:49	Súbor JPG	24 kB
 pes2	2. 12. 2020 17:14	Súbor JPG	24 kB

ZHRNUTIE A VYLEPŠENIE

Program je spustiteľný a funguje. Dala by sa ale vylepšiť metóda posielania – nie všetko naraz ale postupne po balíkoch. Môj program funguje na kratšie správy a menšie súbory, pretože pri veľkých súboroch a správach sa presiahne veľkosť bytearray, ktorá je 256. Ďalej mi funguje keep alive iba na začiatku, kde kontroluje, či je klient vôbec zapnutý a ak sa neozve do nejakého času, tak sa vypne aj server. Dalo by sa to vylepšiť, že bude kontrolovať aj pri načítavaní zo vstupu u klienta – ak klient dlho nič nezadá čo chce robiť, server by sa vypol.