Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Tibor Vanek

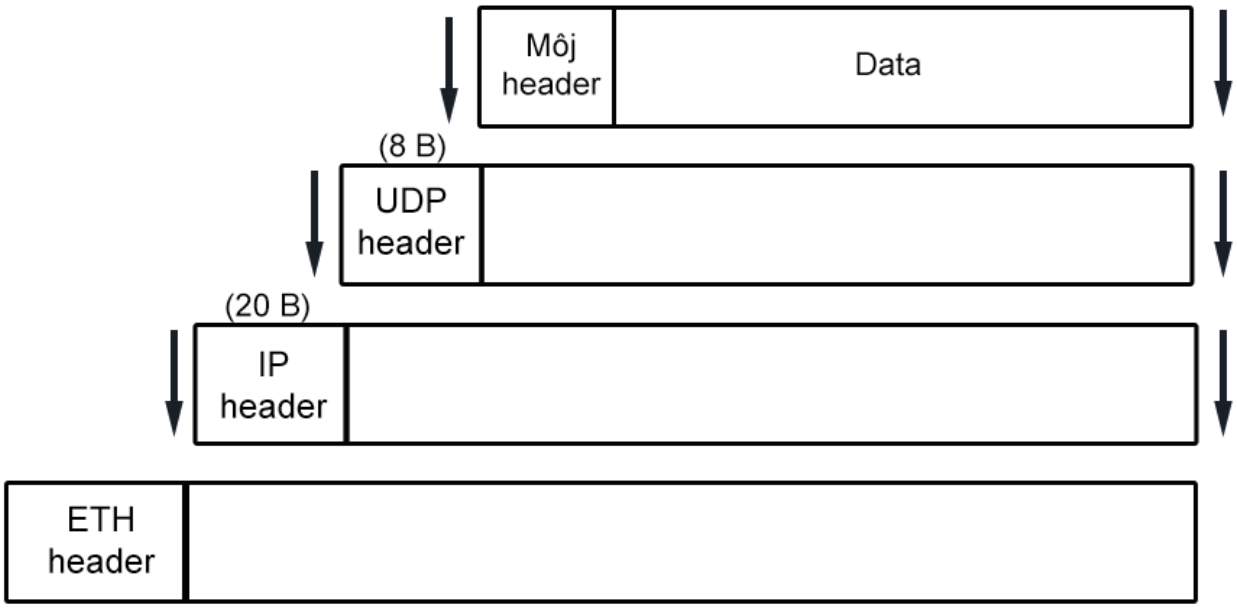
Zadanie 2 – Komunikácia s využitím UDP protokolu

Počítačové a komunikačné siete

Predmet: Počítačové a komunikačné siete
Čas cvičenia: Piatok 8:00
Cvičiaci: Ing. Matej Janeba 2021/2022

Návrh

Protokol bude vnorený v protokole UDP nasledovne:



Moja hlavička

Typ	Veľkosť	Počet fragmentov	CRC	Data
1 B	2 B	2 B	2 B	

Dokopy **7 B**

Typy:

Typ	Význam
0	Inicializácia
1	Prenos dát (správa)
2	Prenos dát (súbor)
3	Poškodené dáta
4	Keep-alive

maximálna veľkosť fragmentu je 1500 – IP hlavička (20) – UDP hlavička (8) – moja hlavička (7) == **1465 B**

Implementáciu plánujem vykonať v jazyku Python kvôli jeho užitočným funkciám. Na začiatku bude mať program možnosť, či má pracovať ako server alebo klient. Ak bude klient, používateľ zadá IP servera, port a veľkosť fragmentu a správu na odoslanie. Následne prebehne inicializácia, t. j. pošle sa inicializačný paket na server a keď príde ACK, začnú sa posielať dáta. Ak je celková veľkosť správy > nastavená veľkosť na fragmentáciu, správa sa rozdelí na fragmenty a pošle sa jednotlivo. Po každom fragmente sa na serveri overí checksum paketu a ak sedí, odošle sa klientovi potvrdenie ACK a klient pošle ďalší paket. Takto sa to opakuje, kým sa nepošle celá správa / súbor.

Ak sa program spustí ako server, používateľ nastaví port na ktorom bude server počúvať a server sa spustí. Keď príde inicializačný paket, nastavia sa určité parametre a odošle sa ACK. Následne sa prijímajú pakety, dokým sa neprijmú všetky. Každý paket sa po prijatí kontroluje kvôli zvolenej ARQ metóde stop-and-wait. Ak príde poškodený paket, odošle sa žiadosť klientovi o poslanie znova. Po úspešnom prijatí celej správy server čaká 18 sekúnd na keep-alive signál od klienta. Ak nepríde, zatvorí sa komunikácia na porte sa čaká sa na nové inicializovanie.

Plánujem zahrnúť aj možnosť prepnutia funkcie programu z servera na klient a opačne. Klient bude môcť prepnúť keď skončí posielanie. Server bude môcť prepnúť keď vyprší keep-alive timeout.

Checksum metóda

Checksum slúži na overenie nepoškodenosti poslanej správy. Ja budem používať CRC 16 z knižnice *binascii* v Pythone. Funguje na princípe XOR-ovania a posúvania do prava – pod prvú jednotku z ľava.

Polynóm crc16:

$x^{16} + x^{12} + x^5 + 1$

ARQ metóda

Plánujem použiť ARQ metódu **stop-and-wait**.

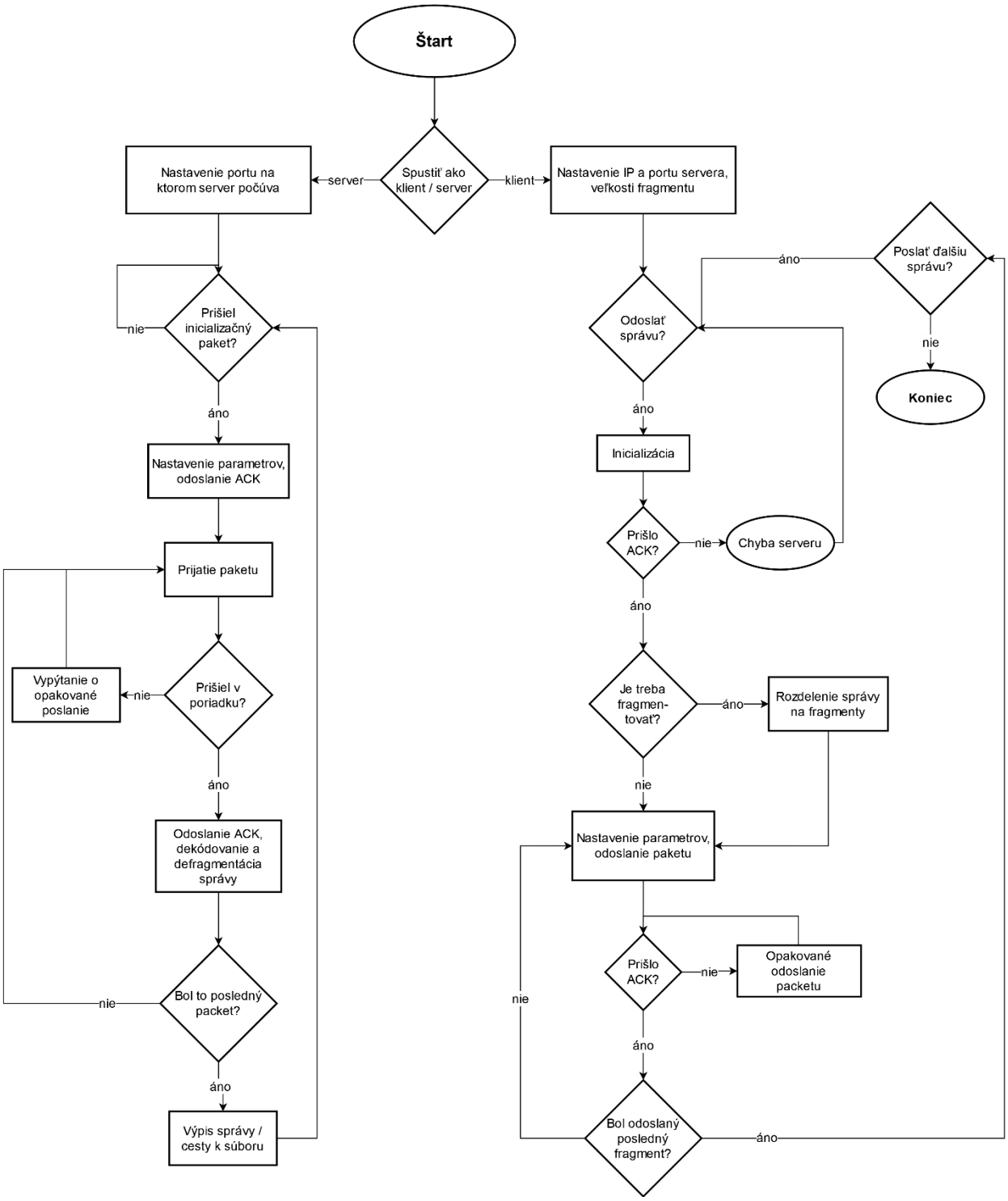
Výhoda je v tom, že pakety prídu vždy v správnom poradí, keďže sa po odoslaní každého fragmentu čaká na potvrdenie od servera. Jedna nevýhoda tejto metódy je, že ak sa správa ACK od servera po ceste stratí, klient si bude myslieť, že server jeho správu neprijal a pošle ju znova. Toto by spôsobovalo problém duplicitnosti, ktorý ale v tomto zadaní netreba riešiť, keďže **chyba** je simulovaná **len v dátovej časti** paketu. Nevýhoda tejto metódy je to, že ak klient odošle **veľa chybných paketov**, efektivita sa zníži.

Ak do určitého časového limitu nepríde klientovi ACK od servera, pošle tento paket znova. Časový limit sa resetuje po každom úspešne odoslanom fragmente.

Keep-Alive

Po inicializovaní spojenia bude musieť klient každých 15 sekúnd posielať na server keep-alive správu. Ak na server nepríde keep-alive správa do jeho timeout-u 18 sekúnd, spojenie bude zrušené.

Sekvenčný diagram



Finálne odovzdanie

Zmeny oproti návrhu

Pridal som **nový typ správy** v hlavičke – „3“ (***ACK***). Typy *Poškodené dáta* som posunul na číslo 4 a *Keep-alive* som odstránil, keďže teraz nezáleží, aký typ príde na server.

Aktualizované typy:

Typ	Význam
0	Inicializácia
1	Prenos dát (správa)
2	Prenos dát (súbor)
3	Prenos ACK
4	Poškodené dáta

Zmenil som navrhnuté fungovanie **keep-alive**. Teraz sa po vykonaní programu v roli *klient* alebo *server* ukáže užívateľské menu, v ktorom sa dá vybrať, či chce používateľ pokračovať a v akej roli. Takto som vyriešil aj **výmenu rolí**. Ak po inicializácii medzi serverom a klientom prestane klient odosielať (preruší sa spojenie alebo nastane iná chyba) a tým pádom na server nepríde žiadny paket, server má 10 sekúnd **timeout interval**, ktorý keď vyprší, tak sa zatvorí spojenie.

Prejdenie na užívateľské menu po vykonaní funkcie sa líši aj od sekvenčného diagramu z návrhu. Po odoslaní posledného fragmentu už program nezostáva v rovnakej funkcii.

Dodatočný opis riešenia

Na zhrnutie opisu z návrhu popíšem funkcie programu. V užívateľskom rozhraní je na začiatku možnosť spustiť program ako *klient* / *server*. Oba majú rozdielne inicializácie vo svojich funkciách.

Inicializácia serveru prebieha vo funkcii `server_init()`. Používateľ vloží len port serveru a čaká na spojenie. Ako inicializačný paket sa tu používa len string `"0"`, keďže je to v tomto prípade jediná časť hlavičky, ktorú je treba poslať (ostatné polia by boli prázdne). Klient takýto paket pošle serveru a server odpovie rovnako. Úspešné vykonanie tohto signalizuje bezproblémové spojenie. Následne program prechádza do funkcie `server_receive_message()`, v ktorej sa prijímajú fragmenty správy a spájajú. Po prijatí fragmentu sa vypočíta CRC hodnota dátovej časti a porovná s hodnotou od odosielateľa. Ak sa nerovnajú, server si vyžiada opakované poslanie konkrétneho fragmentu. Po úspešnom prijatí fragmentu sa vypíše správa s jeho veľkosťou a číslom fragmentu. Ak pri prenose správy nepríde žiadny paket serveru do 10 sekúnd, vyprší timeout

inerval a spojenie sa ukončí. Ak bol prijatý **súbor**, tak **prvý fragment** vždy obsahuje len **názov súboru**. Všetky ostatné obsahujú jeho dáta.

Inicializácia klienta prebieha vo funkcii `client_init()`. Používateľ vloží IP adresu a port, na ktorom server počúva, veľkosť fragmentov a typ správy (správa alebo súbor). Po nadviazaní spojenia prechádza do funkcie `client_send_message()`. Ak je treba, tak sa dáta fragmentujú a vložia do listu, z ktorého sa budú jednotlivo posielať. Ak je posielaný súbor, prvý fragment sa umelo vytvorí a pripoja sa k nemu fragmenty dát. Fragmenty sa v cykle po jednom posielajú, pričom sa pred túto dátovú časť ešte pridáva vlastná hlavička. Inak klient klasicky spracúva fragmenty, dokým nie sú úspešne prenesené všetky. V tejto funkcii sa dá využiť **možnosť poškodenia dátovej časti** niektorého fragmentu, ak je treba.

Zobrazenie prenosu

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	33	60399 → 5005 Len=1
2	0.000355	127.0.0.1	127.0.0.1	UDP	33	5005 → 60399 Len=1
3	0.000570	127.0.0.1	127.0.0.1	UDP	54	60399 → 5005 Len=22
4	0.102695	127.0.0.1	127.0.0.1	UDP	33	5005 → 60399 Len=1
5	0.103010	127.0.0.1	127.0.0.1	UDP	1039	60399 → 5005 Len=1007
6	0.103405	127.0.0.1	127.0.0.1	UDP	33	5005 → 60399 Len=1
7	0.103683	127.0.0.1	127.0.0.1	UDP	1039	60399 → 5005 Len=1007
8	0.104086	127.0.0.1	127.0.0.1	UDP	33	5005 → 60399 Len=1
9	0.104364	127.0.0.1	127.0.0.1	UDP	1039	60399 → 5005 Len=1007
10	0.104750	127.0.0.1	127.0.0.1	UDP	33	5005 → 60399 Len=1
11	0.105020	127.0.0.1	127.0.0.1	UDP	1039	60399 → 5005 Len=1007
12	0.105407	127.0.0.1	127.0.0.1	UDP	33	5005 → 60399 Len=1
13	0.105667	127.0.0.1	127.0.0.1	UDP	183	60399 → 5005 Len=151
14	0.106073	127.0.0.1	127.0.0.1	UDP	33	5005 → 60399 Len=1

Prenos súboru veľkosti 4144 B s veľkosťou fragmentu 1000. Prvé 2 pakety sú inicializačné. Tretí paket obsahuje názov súboru, potom nasledujú dáta. Medzi nimi sa posiela ACK od servera klientovi.

Cesty k súboru na jednotlivých uzloch:

Server:
`D:/Škola STU/Python projects/PKS_Zadanie2/receiving_files/mediumSubor.txt`

Klient:
`D:/Škola STU/Python projects/PKS_Zadanie2/sending_files/mediumSubor.txt`

Používateľské rozhranie:

Server

```
---------------UDP Komunikator---------------
----------ZACIATOK VYBERU ROLE----------
Ak chcete skoncit program, vlozte '0'
Ak chcete program spustit ako CLIENT, vlozte '1'
Ak chcete program spustit ako SERVER, vlozte '2'
Vasa volba: 2
Zvolili ste si server
Zadajte server port: 5005
Server caka na spojenie...

Server received initialization request
from address: ('127.0.0.1', 61070)
ACK for init sent to client!

Prijaty fragment cislo 1 / 1
Velkost fragmentu (bez hlavicky): 14 B

Na server prisla sprava: b'Vzorova sprava'
dlzka spravy:  14
Dekodovana sprava: Vzorova sprava
```

Klient

```
---------------UDP Komunikator---------------
----------ZACIATOK VYBERU ROLE----------
Ak chcete skoncit program, vlozte '0'
Ak chcete program spustit ako CLIENT, vlozte '1'
Ak chcete program spustit ako SERVER, vlozte '2'
Vasa volba: 1
Zvolili ste si client
Zadajte server IP: 127.0.0.1
Zadajte server port: 5005
Zadajte velkost fragmentu (2 - 1465): 1000
Ak chcete poslat SPRAVU, vlozte '1', ak chcete poslat SUBOR, vlozte '2'
Zvoleny typ: 1
Napiste spravu: Vzorova sprava
waiting for confirmation...
Accepted confirmation for init by client

Odoslany fragment cislo 1 / 1
Velkost fragmentu (bez hlavicky):  14 B
Cakanie na ACK... ACK!

Vsetky fragmenty spracovane klientom
```

Použité knižnice:

- socket
- time
- math
- os
- binascii