**Slovenská technická univerzita v Bratislave**

Fakulta informatiky a informačných technológií

**Počítačové a komunikačné siete**

Zadanie 2: Komunikácia s využitím UDP protokolu

Richard Šléher

2023/24

**Obsah**

[Voľba jazyka a knižníc 2](#_Toc152597711)

[Návrh riešenia 2](#_Toc152597712)

[Štruktúra hlavičky 3](#_Toc152597713)

[Diagram fungovania programu 4](#_Toc152597714)

[Priebeh funkcie switch 5](#_Toc152597715)

[Zvolená metóda CRC 5](#_Toc152597716)

[Funkcia keep-alive 6](#_Toc152597717)

[Stop & wait ARQ 6](#_Toc152597718)

[Sekvenčné diagramy hlavných funkcionalít 7](#_Toc152597719)

[Zachytenie komunikácie vo Wiresharku 10](#_Toc152597720)

[Záver a zmeny 12](#_Toc152597721)

Voľba jazyka a knižníc

Program implementujem v programovacom jazyku Python. Dôvod je jednoduchosť a väčšie množstvo knižníc v porovnaní s inými jazykmi. Program je písaný v prostredí Visual Studio Code, ktoré je ľahko upravovateľné. Posielanie dát bude zachytávané programom Wireshark. Všetky knižnice, ktoré používam v programe sú built-in okrem *numpy*, ktorú je treba nainštalovať. Použité knižnice: *socket* na komunikovanie medzi uzlami*, os* na prácu s cestami k súborom*, threading* na vytvorenie vlákna na udržiavania spojenia*, binascii* pre výpočet CRC*, time* na uspanie programu pri posielaní keep-alive paketov*, math* na zaokrúhlenie*, struct* na tvorbu vlastnej hlavičky.

Návrh riešenia

Hlavná časť programu je funkcia main(), ktorá slúži ako centrálna časť a umožňuje užívateľovi vybrať si medzi rôznymi režimami programu ako je klient, server alebo ukončenie programu.

Klientova časť obsahuje funkcie client\_setup() a client(). Funkcia client\_setup() je určená na nastavenie klienta, získanie adresy a portu servera a následne sa pokúsi pripojiť na server. Funkcia client() obsahuje hlavnú logiku klienta, ktorá umožňuje užívateľovi vybrať si medzi rôznymi úlohami ako je odoslanie správy alebo súboru, prepnutie do režimu serveru alebo ukončenie spojenia. Taktiež má možnosť naslúchať príkazom zo serveru.

Serverová časť programu obsahuje funkcie server\_setup() a server(). Funkcia server\_setup() slúži na nastavenie servera, získanie portu, na ktorom bude server počúvať a následne spustí server a čaká na pripojenie klienta. Funkcia server() obsahuje hlavnú logiku pre server, ktorá po pripojení klienta prijíma a spracováva dáta od klienta a obsahuje konzolu pre rôzne funkcie servera ako je odoslanie správy klientovi, prepínanie režimov alebo ukončenie spojenia.

Okrem týchto hlavných funkcií existujú ďalšie pomocné funkcie pre tvorbu a spracovanie hlavičiek správ, ako aj funkcie na vypočítanie CRC hodnoty. Ďalej sú v programe použité funkcie, ktoré bežia na vláknach ako napríklad funkcie keeping\_alive() a server\_console().

Štruktúra hlavičky

*Tabuľka 1* znázorňuje políčka a ich veľkosti v implementácii. Políčko FLAG má záznam o type správy. Všetky typy správ sú v *Tabuľka 2*. PACKET NUM obsahuje číslo poslaného datagramu. CRC je políčko na kontrolu či sa správa poslala bez chyby. Veľkosť políčka DATA sa mení používateľovým vstupom. Maximálna dĺžka je 1467B, lebo treba odčítať hlavičky nižších protokolov a mojej implementovanej hlavičky.

|  |  |  |  |
| --- | --- | --- | --- |
| 1B | 2B | 2B | 0 – 1467B |
| FLAG | PACKET NUM | CRC | DATA |

Tabuľka 1 – Hlavička protokolu

|  |  |
| --- | --- |
| FLAG | FUNKCIA |
| 1 | Inicializácia |
| 2 | Keep-alive |
| 3 | Dáta |
| 4 | ACK |
| 5 | NACK |
| 6 | Posledný paket (ukončenie posielania) |
| 7 | Prepnutie (Switch) |
| 8 | Ukončenie spojenia |

Tabuľka 2 – Flagy a ich funkcie

Diagram fungovania programu

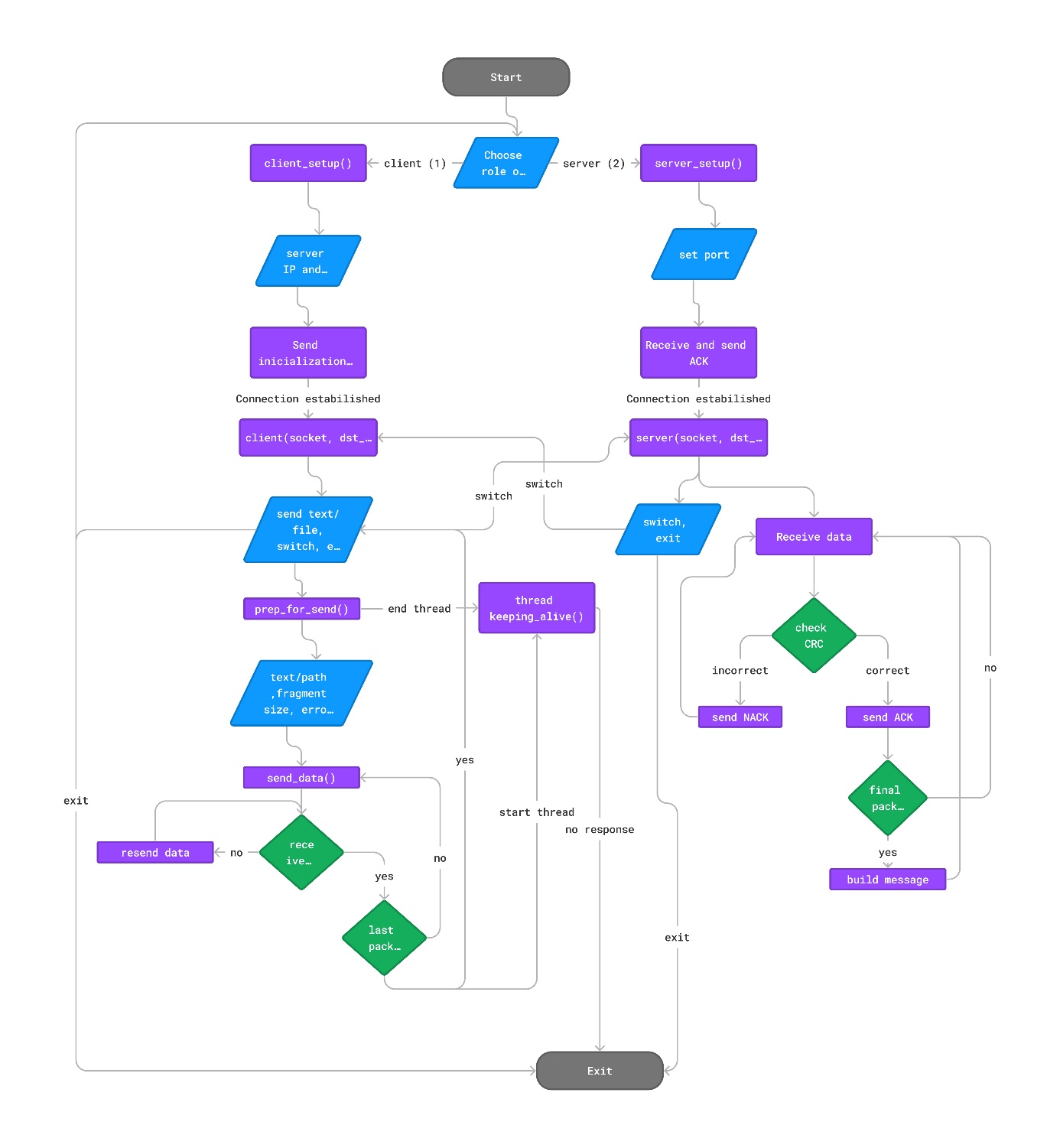


Diagram 1 – Fungovanie programu

Priebeh funkcie switch

Táto funkcia má za úlohu prepnúť funkcionality klienta a servera. Táto možnosť funguje z oboch strán komunikácie ako z klienta tak zo servera.

Priebeh prepnutia z klienta je jednoduchší, keďže server stále počúva a klient mu môže hocikedy poslať správu o žiadosť o prepnutie. Funkcionalita funguje tak, že klient pošle správu s FLAG-om nastaveným na 7. Server túto správu prijme a pošle naspäť potvrdenia s rovnakým FLAG-om, následne zavrie svoj socket a zavolá funkciu client() s argumentami socket a cieľová adresa. Klient spraví to isté s tým rozdielom, že zavolá funkciu server().

Zo strany servera je nutné nastaviť na klientovi mód *listen,* ktorý nastaví na klientovi počúvanie na 15 sekúnd. V tomto časovom okne môže server poslať správu s žiadosťou o prepnutie. Ďalej sa vykoná všetko tak isto ako pri výmene zo strany klienta.

Zvolená metóda CRC

Slúži na kontrolu, či sa dané dáta poslali korektne v správe. Aby sa mohla poslať správa o znovu poslaní alebo nie. Na výpočet CRC hodnoty používam funkciu *crc\_hqx* z knižnice *binascii*, ktorá je built-in v Pythone. Zvolil som ju, lebo zisťuje hodnotu CCITT, ktorý používa polynóm x16 + x12 + x5 + 1. Pre toto zadanie stačí 16-bit-ové CRC, lebo má dostatočnú presnosť pri veľkostiach, s ktorými budeme pracovať.

Postup pri výpočte CRC-16 CCITT sa používa polynóm 0x1021, čo zodpovedá polynómu x16 + x12 + x5 + 1 v binárnom tvare. Následne sa každý bit vstupných dát postupne XORuje s najvyššími 16 bitmi hodnoty CRC. Ak je výsledok XOR operácie 1, do hodnoty CRC sa pridá hodnota generujúceho polynómu 0x1021. Proces sa opakuje pre každý bit vstupných dát, až kým nie sú spracované všetky bity. Výsledná hodnota CRC je potom pripojená k pôvodným dátam a odoslaná spolu s nimi. Na strane prijímateľa sa vykoná rovnaký výpočet a porovná sa s poslaným ak sa zhodujú tak boli dáta korektne poslané inak sa pošle správa o opätovnom poslaní.

Funkcia keep-alive

Funkcia keeping\_alive() posiela periodicky správy s FLAG-om *keep-alive* na server a monitoruje jeho odpovede, aby sa uistila, že spojenie nebolo prerušené. Po neúspešnom pokuse o udržanie spojenia sa navyšuje hodnota s neúspešnými pokusmi a takisto sa navyšuje doba čakania na odpoveď. Táto hodnota sa každým neúspešným pokusom navyšuje o 5 sekúnd. Taktiež sa navyšuje čas za ktorý sa znova pošle správa *keep-alive* na server. Ak sa na server nepodarí pripojiť 3-krát po sebe (5s, 10s, 15s) tak sa spojenie a program skončia.

Stop & wait ARQ

Táto metóda funguje na princípe, kde sa pošle rámec s dátami a klient čaká na odpoveď vo forme ACK alebo NACK. Ak príde správa s ACK tak sa pokračuje ďalej a pošlú sa ďalšie dáta. V prípade, že príde NACK sa pošlú rovnaké dáta znova a ak od serveru nepríde žiadna správa pošlú sa dáta znova a čaká sa 15 sekúnd na odpoveď. V prípade, že server neodpovie ani druhý raz predpokladá sa, že sa pripojenie prerušilo a program sa ukončuje.

Výhoda tejto metódy je spoľahlivosť a jednoduchosť implementácie a nevýhoda je vyššia latencia z dôvodu posielania odpovedi na každý prijatý rámec.

Sekvenčné diagramy hlavných funkcionalít

Na sekvenčných diagramoch sú zobrazené priebehy rôznych funkcionalít programu ako napríklad udržiavanie spojenia, posielanie dát a podobne.

Obrázok, na ktorom je text, rad, snímka obrazovky, diagram

Automaticky generovaný popis

Diagram 2 – Prvotná inicializácia spojenia

Obrázok, na ktorom je text, rad, snímka obrazovky, diagram

Automaticky generovaný popis

Diagram 3 – Ukončenie programu

Obrázok, na ktorom je text, rad, diagram, snímka obrazovky

Automaticky generovaný popis

Diagram 4 – Zámena fukncionalít

Obrázok, na ktorom je text, rad, snímka obrazovky, rovnobežný

Automaticky generovaný popis

Diagram 5 – Priebeh posielania dát aj s chybou

Obrázok, na ktorom je text, rad, diagram, rovnobežný

Automaticky generovaný popis

Diagram 6 – Funkcionalita na udržanie spojenia

Zachytenie komunikácie vo Wiresharku

Vytvoril som si vlastný LUA skript na jednoduchšie sledovanie mojich paketov, ktoré som posielal a na jednoduchšiu analýzu jednotlivých políčok. Na *Obrázok 1* je možné pozorovať vľavo dole jednotlivé políčka vlastného protokolu a v strede poslané pakety. Toto je príklad, kde som posielal slovo „Ahoj“ po jednom byte. Zvyšné pakety zobrazujú funkcionalitu udržiavania spojenia so serverom. Obrázok 2 zobrazuje ukončenie komunikácie, kde server neodpovedal na správu keep-alive 3x po sebe (posledné 3 pakety). Obrázok 3 zobrazuje poslanie prvého paketu, kde políčko Data prenáša názov súboru, preto je rozdielnej veľkosti ako ostatné. Prvé dva pakety na obrázku sú na inicializáciu komunikácie.

Obrázok, na ktorom je text, elektronika, snímka obrazovky, softvér

Automaticky generovaný popis

Obrázok 1 – Scenár s poslaním dát

Obrázok, na ktorom je text, snímka obrazovky, displej, softvér

Automaticky generovaný popis

Obrázok 2 – Zobrazenie ukončenia, keď server neodpovedá

Obrázok, na ktorom je text, elektronika, snímka obrazovky, displej

Automaticky generovaný popis

Obrázok 3 – Zobrazenie prenosu mena súboru

Záver a zmeny

Na záver by som chcel porovnať, čo sa zmenilo vo finálnom programe oproti návrhu a odôvodiť moje rozhodnutia implementácie jednotlivých bodov zadania.

V hlavičke som odstránil jedno pole s názvom TYPE. Toto uchovávalo o aký typ správy sa jedná (text alebo súbor). Bolo zbytočné lebo názov súboru posielam v prvom poslanom pakete a z toho vie program zistiť, že sa jedná o súbor.

Ďalej som trocha upravil keep-alive metódu. Tu som pridal podmienku na ukončenie spojenia vtedy, keď keep-alive správa nepríde 3x po sebe od servera. Pričom sa okno na prijatie správy zväčšuje o 5 sekúnd (5, 10, 15). Pred tým to bola jedna správa po 10 sekundách.

CRC metódu som takisto zmenil z 32 bitovej na 16 bitovú, lebo mi zaberala viac miesta v hlavičke a nebol tam žiadny rozdiel pri používaní 16 bitovej.

ARQ metóda sa takisto zmenila. Na začiatku som mal v pláne použiť selective repeate, ale zostal som pri stop & wait z dôvodu jednoduchšej štruktúry a zrejeme aj jednoduchšej doimplementácie nových prvkov do kódu.