Komunikácia s využitím UDP protokolu - Dokumentácia

Fakulta Informatiky a Informčaých technológií Slovenskej Technickej Univerzity

Semestrálny projekt z PKS

Ondrej Krajčovič

xkrajcovico@stuba.sk

24.Novembra 2024

Zadanie

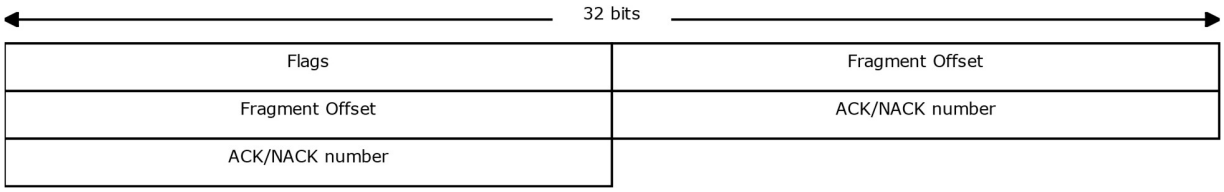
[1]Cieľom zadania je navrhnúť a implementovať P2P aplikáciu využívajúcu vlastný protokol postavený nad UDP (User Datagram Protocol) v transportnej vrstve sieťového modelu TCP/IP. Aplikácia bude umožňovať komunikáciu dvoch účastníkov v lokálnej Ethernet sieti, vrátane výmeny textu a prenosu ľubovoľných súborov medzi počítačmi (uzlami). Oba uzly budú fungovať súčasne ako prijímač aj odosielateľ.

Výstupom tejto časti je dokumentácia, v ktorej predstavíte návrh svojho protokolu a plán implementácie jednotlivých mechanizmov. Následne túto dokumentáciu prediskutujete so svojím cvičiacim počas kontrolného bodu, kde získate pripomienky na zapracovanie do praktickej časti. Dokumentácia musí obsahovať nasledujúce časti:

- štruktúru hlavičiek vášho protokolu,
- opis metódy na kontrolu integrity prenesenej správy (napr. ak si zvolíte CRC16, tak ho opíšete ako sa vypočíta, rovnako postupovať pri iných metódach),
- opis metódy na zabezpečenie spoľahlivého prenosu dát (ARQ),
- opis metódy na udržanie spojenia (Keep-Alive),
- diagramy opisujúce predpokladané správanie vášho uzla/uzlov, použite UML diagramy ako napr. sekvenčný, aktivity a stavový (používajte vhodné nástroje, ako napr. miro alebo draw.io, nie skicár).

[1] - github

Hlavička



Obr.1:schéma hlavičky protokolu

*Hlavička protokolu má dĺžku **10 bajtov**, resp 80 bitov. Hlavička sa skladá z nasledovných údajov:*

flags 16 bitov / 2B

typy flag-ov:

- FLAG_SYN: binárne: 10000000: Začiatok spojenia
- FLAG_ACK: binárne: 01000000: Potvrdenie prijatia
- FLAG_NACK: binárne: 00100000: Negatívne potvrdenie prijatia
- FLAG_FIN: binárne: 00010000: Ukončenie spojenia
- FLAG_DATA: binárne: 00001000: Dáta v správe
- FLAG_FRAGMENTED: binárne: 00000100: Správa je fragmentovaná
- FLAG_LAST_FRAGMENT: binárne: 00000010: Posledný fragment správy
- FLAG_KEEPALIVE: binárne: 00000001: Udržiavanie spojenia

Kontrolné Správy:

- SYN_MSG: FLAG_SYN
- SYNACK_MSG: FLAG_SYN | FLAG_ACK
- ACK_MSG: FLAG_ACK
- NACK_MSG: FLAG_NACK
- FIN_MSG: FLAG_FIN
- FINACK_MSG: FLAG_FIN | FLAG_ACK
- KEEPALIVE_MSG: FLAG_KEEPALIVE

Dáta Správy:

- TEXT_MSG: FLAG_DATA
- TEXT_MSG_FRAG: FLAG_DATA | FLAG_FRAGMENTED
- TEXT_MSG_FRAG_L: FLAG_DATA | FLAG_FRAGMENTED | FLAG_LAST_FRAGMENT
- FILE_MSG: FLAG_DATA | FLAG_FRAGMENTED
- FILE_MSG_L: FLAG_DATA | FLAG_FRAGMENTED | FLAG_LAST_FRAGMENT

fragment Offset (fragNumber) 32 bitov/4B

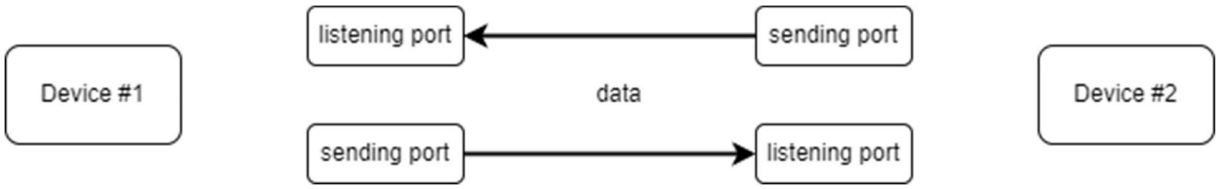
Identifikačné číslo aktuálneho fragmentu správy. Ak správa nie je fragmentovaná, hodnota je 0.

ACK/NACK number (ackNumber) 32 bitov/4B

Číslo potvrdenia prijatia predchádzajúcej správy.

2B: Checksum:

Štyri bajty obsahujúce „checksum“ headeru a dát kalkulovaných pomocou vlastného CRC16, určený na verifikáciu obsahu packetu. Nie je plne vsadený do hlavičky, pripája sa k správe po vytvorení packetu, a to na jeho koniec



Obr2.: schéma komunikácie medzi dvoma bodmi

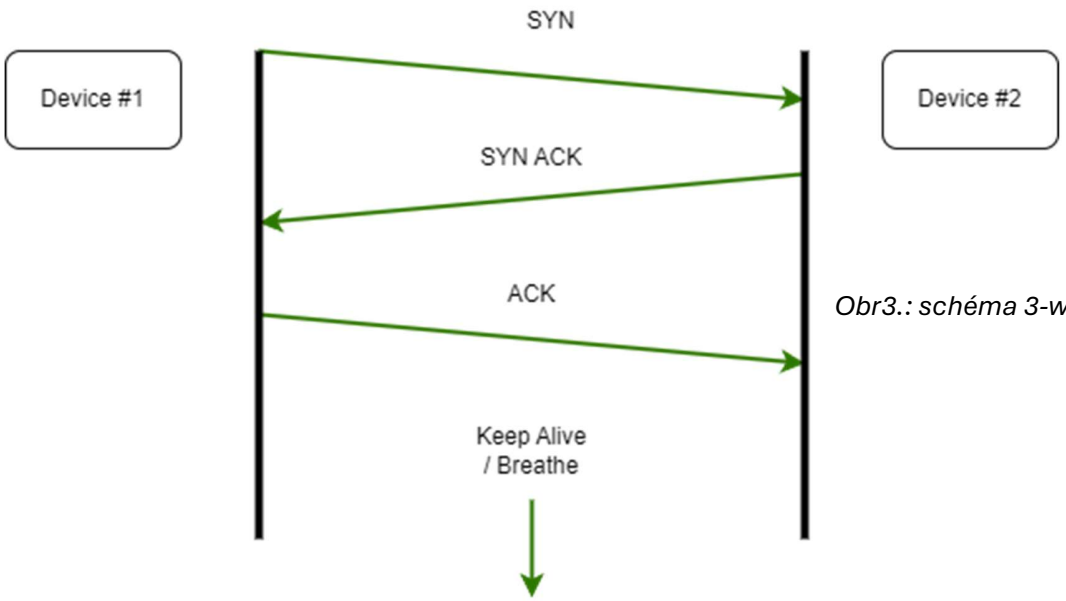
Poznámka: Fragment offset sa používa na strane odosielateľa a ACK/NACK number sa používa na spätnom chode smerom od prijímateľa. Je to tak z dôvodu prehľadnosti, nakoľko sa mne osobne ako dyslektikovi nedalo rozoznať ktorou stranou ide packet v prípade že som mal iba fragment offset. Ide tak síce o istú neefektivitu, ale bohužiaľ bola nevyhnutná

Nadviazanie spojenia - Handshake

Na nadviazanie spojenia plánujem použiť TCP inšpirovaný three way hadnshake. Na jeho utilizáciu som pri finálnej verzií spravil vlastný thread

1. Proces začína tým, že Point#1 vyšle SYN segment so žiadosťou o spojenie zároveň s inicializačným sequence number(SQN).
2. Point#2 odpovie segmentom SYN-ACK na potvrdenie prijatia
3. Point#1 následne potvrdí spojenie zaslaním ACK segmentu.

Inicializácia bude prebiehať dvakrát, a to z dôvodu že každá osoba bude mať vlastný sending port a listening port.



Obr3.: schéma 3-way handshake-u

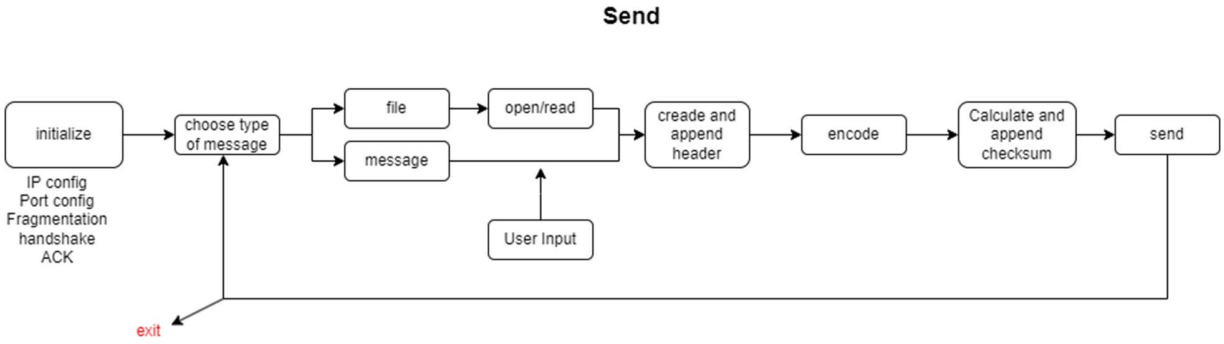
Udržiavanie spojenia, posielanie packetov a ukončenie spojenia

Na udržanie spojenia, inicializovaného handshake-om bude každým kanálom v pravidelných intervaloch(5s) poslianý packet bez dát, s keep alive flag, v prípade že po časový interval 15s jedno zariadenie neobdrží daný packet, bude komunikácia následne zrušená.

Pre posielanie packetov bude každým odoslaním otvorený vopred dohodnutý port odosielatľom, následne bude správa odoslaná a po prijatí spätného Acknowledgement packetu bude komunikácia zo strany odosielateľa dočastne zastavená

pri ukončovaní spojenia sa postupuje nasledovne(ide o vlasntú variáciu inšpirovnanú TCP four-way handshake-om):

odosielateľ pošle FIN Peer B.
Prijímateľ prijme FIN a pošle späť FINACK.
Prijímateľ môže tiež poslať svoj vlastný FIN Peer A, ak sa ešte nerozhodol ukončiť spojenie.
odosielateľ prijme FINACK (a prípadne ďalší FIN od Peer B) a dokončí ukončenie tým, že uzavrie spojenie.



Obr4.: schéma procesu odosielania správy

Fragmentácia

Pre zníženie dĺžky jednotivých odoslaných packetov budú správy posielané v častiach (ďalej fragmentoch). Správy teda budú postupne odosielané zo send portu odosielateľa v packetoch o vopred určenej veľkosti. Postup je teda nasledovný. Správa sa postupne načíta do packetu, pridá sa header a checksum. Následne sa packet odošle. Po prijatí packetu sa na strane prijateľa správa rozdelí porovná sa checksum a prípadne sa pripojí ku zvyšku doteraz prijatej správy.

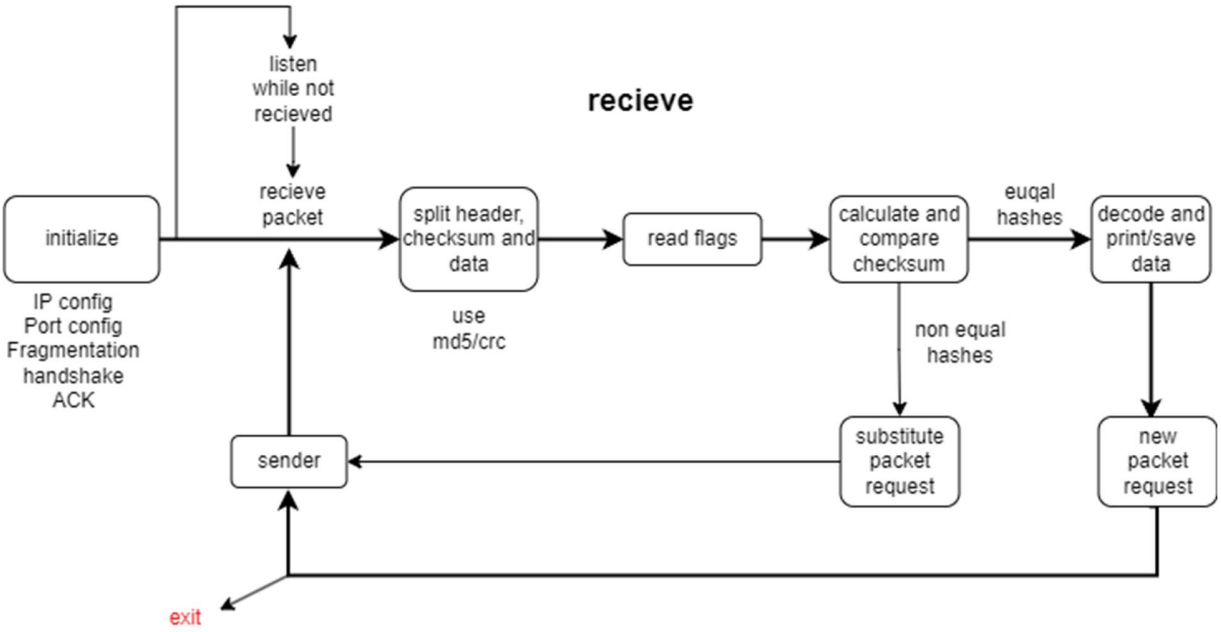
Overovanie korektnosti a prijímanie packetov

Pred odoslaním správy sa z hlavičky a dát vytvorí checksum pomocou CRC16, ktorý bude následne vložený za dáta. Po prijatí packet bude správa rozdelené naspäť na hlavičku checksum a dáta. Následne sa opäť vyráta checksum z prijatých údajov a porovná sa s prijatým checksumom. Ak sa rovnajú správa bola neporušená a prijmateľ odošle Acknowledgement o prijatí správy resp, žiadosť o ďalší packet. Ak sa však prijatý checksum a vykalkulovaný checksum nerovnajú prijatý packet sa zadodí, a žiada sa o daľší, nahrádný packet.

Funkcia **CRC16** funguje nasledovne:
CRC sa nastaví na počiatočnú hodnotu FF FF.
Pre každý bajt b v vstupných dátach sa vykoná XOR operácia medzi aktuálnym CRC a bajtom posunutým o 8 bitov doľava (b << 8).
Následne pre každý z 8 bitov bajtu program skontroluje, či posledný bit (bit 15) CRC je 1.
Ak je tak program vykoná bit-shift CRC o 1 bit doľava a XOR CRC s polynomom poly (štandardne 0x1021).
Ak nie je tak program iba posuhie CRC o 1 bit doľava.
Po každom posunutí program zabezpečí, že CRC zostane 16-bitové pomocou maskovania (CRC &* FF FF)
*logický and

ARQ - zabezpečenie spoľahlivého prenosu dát

Implementovaný bol selective repeat ARQ. Sliding window, mechanizmus ktorý po odoslaní k+n frames prijíma k-ty spätný acknowledgement od reciever-a.v prípade že sa m-tý
(m ∈ <k – k+n>) frame nedostane k odosielateľovi, všetky ďalšie frame-y sa rušia a odosiela sa opäť m-tý frame.



Obr5.: schéma procesu prijímania správ

Keep Alive

Program počas trvania spojenia posiela header správy bez dát, ktorý bude obsahovať keep alive flag. Tento packet sa bude vysielať v pravidelnom intervale 5s, v prípade že ho príjmač nezachytí počas 15s, spojenie sa z našej strany uzavrie a program sa ukončí.

Umelá Chyba

Nakoľko sa pri bežnom prenose dát chybovosť limitne blíži nulovej, pri testovaní budeme musieť chybu simulovať. Bude tak mechanickým pozmenením náhodných 8 bitov checksum-u. Program bude musieť packet s chybným checksum-om zachytiť a poslať spätný NACK, ktorý po dorazení na odosieľatelovu stranu zaháji opätovné odoslanie packetu podla fragment offset čísla

Testovanie

Program som testoval nasledovne:
po nadviazaní spojenia som odoslal textovú správu z oboch strán.
Následne som poslal ~2.2MB file (fotku z detstva), ktorý bol rekonštruovaný behom ~20s.
Po ňom som zadefinoval úmyselnú chybu na packet 57, a následne poslal menší file(screenshot z hry civ).
Po úspešnom obdržaní file-u z úmyselnou chybou som zadal input 5, t.j. fin, program sa nasledovne ukončil s uzatvorením portov.

Okrem ukázaných vlastností umožňuje program nasledovné:

- Zmenu miesta ukladania file-ov (by default ide o aktuálny adresár)
- Zmenu veľkosti posielaných fragmentov (by default ide o 1420B)
- Samo-ukončenie po odpojení jedného pointu komunikácie

Dokumentácie testu bude priložená vo fotkách a priloženej prezentácií

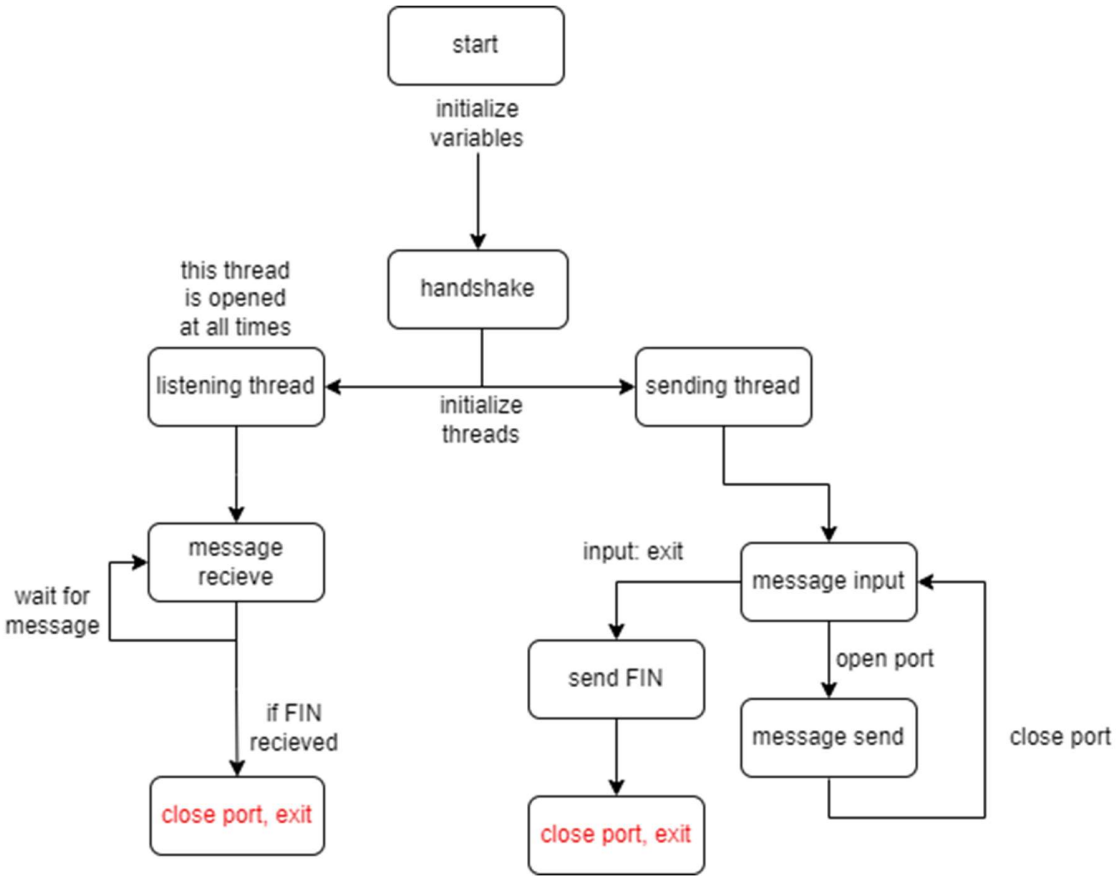
Záver

Pri dohotovení projektu som spravil nasledovné: zefektívnil Štruktúru hlavičky, umožnil fragmemntáciu, kontrolu správnosti údajov pomocou vlastného crc checksum-u, vytvoril menu pre UI a to z dôvodu pohodlnosti používateľa, umožnil zmenu miesta na ukladanie poslaných súborov, umižnil implementáciu umelej chyby a jej následné zachytenie a opätovné odoslanie. Okrem iného som pridal časovač a pravidelné posielanie keep alive packetov, ktoré v prípade násobného neprijatia ukončí pribeh proramu. Program je relatívne efektívny, (aj keď sú pravdepodobne miesta kde by sa dal zefektívniť) bohate však spĺňa časovú podmienku 60-tich sekúnd na poslanie 2MB súboru. Kód programu bol počas implementácie do finálnej verzie výrazne zmenený.

Použité knižnice:

- *socket:* pre potreby sieťového spojenia p2p komunikácie
- *threading:* pre vytváranie stále-aktívnych vlákien na listening/sending port
- *struct:* na kódovanie správ na byty
- *os:* pre možnosť vystúpiť z programu po odoslaní FIN
- *time:* pre časovače
- random: pre poškoďovanie packtu pri úmyselnej chybe

-



Obr.6: všeobecná schéma riešenia

Obsah

Zdroje:

[1.] - zadanie

[2.] - CRC16

[3.] - prednášky

[4.] - design protokolu

[5.] – selective repeat ARQ

Cvičenia: Adrián Ondov; utorok 16:00