

FIIT STU

PKS Zadanie 2 Dokumentácia

Komunikátor

Meno: Jakub Gašparín

Cvičenie: Piatok 8:00-9:40

Cvičiacci: Ing. Miroslav Bahleda, PhD.

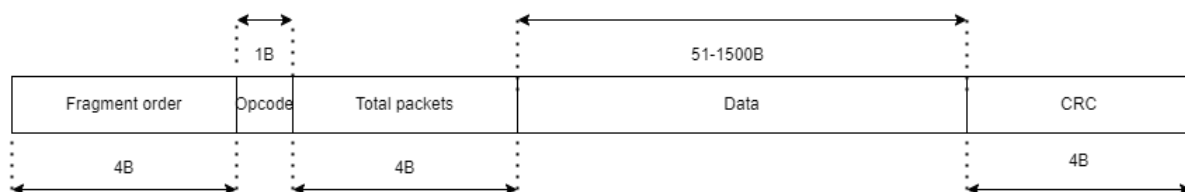
Obsah

1.0 Predpríprava	2
1.1 Návrh hlavičky	2
1.2 Výber typu CRC	3
1.3 Ostatné knižnice	3
1.4 Testovacie súbory	4
1.5 Programovacie prostredie a voľba jazyka	4
2.0 Nadviazanie komunikácie	4
2.1: Prepnutie rolí servera a klienta	6
3.0 Posielanie jednoduchých správ a fragmentácia	6
4.0 Posielanie súborov	11
5.0 Implementácia ARQ metódy	12
6.0 Simulovanie chyby a jej riešenie	14
7.0 Keep-alive metóda	15
8.0 Ukážka mojej komunikácie vo Wireshark	16
9.0 Zmeny medzi finálnou verziou a návrhom	17
10.0 Záver	18

1.0 Predpríprava

1.1 Návrh hlavičky

Návrh mojej hlavičky je nasledovný:



Skladá sa z piatich častí:

- Fragment order: poradie paketu
- Opcode: operácia, ktorú má daný paket vykonať
- Total packets: koľko spolu paketov sa nachádza v danej komunikácii.
- CRC: checksum na kontrolu

Operácie, ktoré sa môžu diať v tejto hlavičke sú nasledovné:

- 0 ACK = Potvrdenie prijatia paketu
- 1 RDY = poslanie správy že klient je pripravený na komunikáciu
- 2 END = ukončenie komunikácie
- 3 WRT = poslanie správy zo vstupu na konzoly
- 4 MPK = oznámenie, že sa posielajú viaceré pakety
- 5 PFL = Poslanie paketu zo súborom
- 6 NCK = Paket nebol správne prijatý a server požiadá o opätovné poslanie
- 7 KAR = Keep-Alive
- 8 SWR = žiadosť o zmenu rolí, t.j. server sa stane klientom a klient sa stane serverom.
- 9 FIN ukončenie komunikácie

Operácie sa v pakete vyznačujú ako ich poradové číslo v tomto zozname. Teda 0 v kóde a v pakete je ACK, 1 je RDY a tak podobne.

Samotná hlavička má minimálnu veľkosť 13B: 4B pre poradie, 1B pre opcode, 4B pre počet celkových paketov poslaných v jednej komunikácii a 4B pre CRC kontrolu. Zvyšné bajty sú alokované pre samotné dáta.

1.2 Výber typu CRC

V práci som použil 16-bitový crc hash. Táto metóda používa 16-bitový crc polynóm:

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

alebo, rozpísaný:

$$1X^{16} + 0X^{15} + 10 + 0X^{14} + 0X^{13} + 1X^{12} + 0X^{11} + 0X^{10} + 0X^9 + 0X^8 + 0X^7 + 0X^6 + 1X^5 + 0X^4 + 0X^3 + 0X^2 + 0X^1 + 1$$

čo sa rovná:

$$1\ 0001\ 0000\ 0010\ 0001$$

pričom musíme odstrániť najvyšší bit zo tohto čísla, teda cifru 1. Z toho nám vznikne nasledovný generátor:

$$1021\ hex$$

S týmto generátorom pracujem knižnica libscrc pre 16-bitový crc hash. Z tejto knižnice som použil funkciu `.buypass(b'X')` na výpočet crc.




1.3 Ostatné knižnice

Knižnice ktoré som ďalej využil v programe sú nasledovné:

- *os* -> kontrola, či existuje cesta k súboru pri posielaní a ukladaní súborov
- *socket* -> nevyhnutnosť na pracovanie s UDP socket programovaním
- *copy.copy* -> v programe využívam možnosť `.append` pre reťazce pri skladaní správ. Pre bezpečnú prácu s dátami sa odporúča v tomto procese použiť `copy.copy`.
- *random* -> simuláciu chyby robím náhodne, napr. keď použijem funkciu `random.randint(1,6)` a vygeneruje číslo 6, tak vložím chybu do paketu.
- *threading* -> musím využiť vlákna na posielanie keep-alive paketov počas behu programu
- *time* -> na meranie času kedy sa má keep-alive paket poslať

1.4 Testovacie súbory

Na testovanie som použil nasledovné súbory:

Icon	Substancia	Typ	Veľkosť
	cviko.docx	1. 12. 2022 18:32	Dokument Micros... 4 492 kB
	files.txt	30. 11. 2022 18:59	Textový dokument 2 kB
	fotka.png	1. 12. 2022 18:25	Súbor PNG 1 288 kB

- *cviko.docx*: dokumentácia zadania z predmetu MIKROP, veľký súbor, vhodný na testovanie spojenia komunikácie
- *files.txt*: obsahuje text pesničky, jednoduchý súbor na testovanie základnej funkcionality programu
- *fotka.png*: fotografia môjho kamaráta použitá s jeho povolením. Otestujem či dokážem preniesť grafické súbory.

1.5 Programovacie prostredie a voľba jazyka

Program bol napísaný v PyCharm 2022.2.2 v jazyku Python 3.10.4. Program je rozdelený do dvoch súborov: *s.py* a *c.py* (server a client respektívne).

2.0 Nadviazanie komunikácie

Komunikácia sa nadväzuje cez UPC socket programovanie. Na to budem potrebovať si vytvoriť dva súbory: *s.py* pre server a *c.py* pre klienta. Pre server sa musím vytvoriť spojenie pre klienta takto:

```

def server():
    global FRAGMENT_SIZE, HOST, PORT
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        HOST = input(f"Current host name is {HOST}, select your new host"
                     f" (or type the same address to keep the current host): \n")
        PORT = int(input(f"Default port of this program is {PORT}, "
                        f"select your new port or keep the same port by typing :"))
        FRAGMENT_SIZE = int(input("Choose fragment size: \n"))
        s.bind((HOST, PORT))
        s.listen()
        while True:
            conn, addr = s.accept()
            with conn:

```

Klient sa spojí so serverom nasledovne:

```

def client():
    i = 0
    global MULTIPLE_FRAGMENTS_FLAG, PACKET_ORDER, OPERATION, FRAGMENT_SIZE, HOST, PORT
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        HOST = input(f"Current host name is {HOST}, select your new host"
                     f" (or type the same address to keep the current host):\n ")
        PORT = int(input(f"Default port of this program is {PORT}, "
                        f"select your new port or keep the same port by typing in the current port val"))
        FRAGMENT_SIZE = int(input("Choose fragment size: \n"))
        s.connect((HOST, PORT))

```

Program poskytuje možnosť si zmeniť IP adresu a port pre pripojenie na server. Ale IP adresu možno dostať aj príkazom `socket.gethostbyname(socket.gethostname())` ktorý nám vráti našu momentálnu IP adresu.

```

# HOST = "192.168.56.1"
HOST = socket.gethostbyname(socket.gethostname())
PORT = 5555

```

Udržanie spojenia sa dá vyriešiť na začiatok veľmi jednoducho a to tak že celé spojenie vložíme do nekonečného while cyklu:

```

while True:
    packetType = input("Are you sending a message or a file or switching roles?\n (msg/file/

    ### MESSAGE #####

    if packetType == "msg":
        OPERATION = WRT
        msg = input("Enter message: ")
        encode_multiple_packets(msg)
        for i in range(len(PACKET_BUFFER)):
            s.send(PACKET_BUFFER[i])

```

2.1: Prepnutie rolí servera a klienta

Prepnutie zabezpečím cez poslanie paketu s operáciou SWR: switch request.

```
def encode_SWR():
    order = 0
    operation = SWR
    total_packets = 1
    msg = "Switch Roles"
    order = order.to_bytes(4, "big")
    operation = operation.to_bytes(1, "big")
    total_packets = total_packets.to_bytes(4, "big")
    msg = msg.encode()
    crc = libscrc.buypass(order + operation + total_packets + msg)
    crc = crc.to_bytes(4, "big")
    packet = order + operation + total_packets + msg + crc
    return packet
```

Tento potom pošlem serveru a spojenie sa ukončí. Nasledovne sa ich role prepnú.

Zo strany klienta:

```
elif packetType == "switch":
    SWR_packet = encode_SWR()
    s.send(SWR_packet)
    s.close()
    server()
```

Zo strany servera:

```
if packet[1] == "_SWR":
    conn.close()
    client()
```

3.0 Posielanie jednoduchých správ a fragmentácia

Ako prvú vec čo som spravil bolo zabezpečenie jednoduchšej komunikácie, t.j. posielanie krátkych správ. Najprv som si ale musel zvoliť veľkosť hlavičky. Ako už viem z môjho návrhu hlavičky, tak všetky jej časti okrem dátovej časti budú vždy zaberat' 13 bytov, takže viem že celková veľkosť mojej hlavičky musí byť minimálne 14 bytov. Prenášať dáta o veľkosti jedného bytu ale nedáva zmysel takže minimálnu veľkosť si zvolím 1024 bitov.

Nasledovne som musel zabezpečiť dynamické určovanie veľkosti fragmentu cez globálnu premennú FRAGMENT_SIZE:

```
FRAGMENT_SIZE = int(input("Choose fragment size: \n"))
```

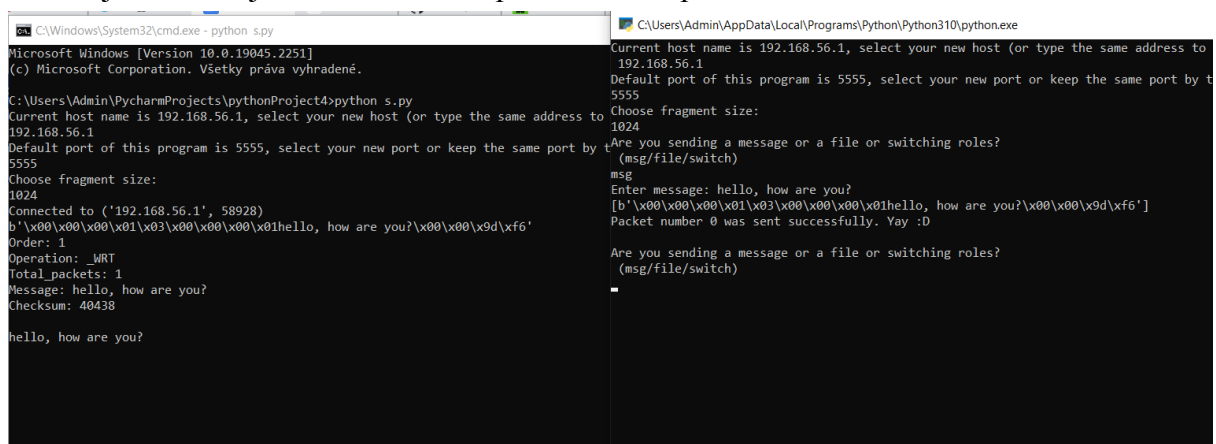
Na toto sa program vždy spýta pred začiatkom každej komunikácie.

Funkcia na tvorbu paketu sa volá encode_multiple_packets() a to preto, lebo už automaticky dokáže rozoznať, či sa jedná o správu ktorá sa zmestí do jedného paketu alebo sa musí rozdeliť na viaceré fragmenty.

```
def encode_multiple_packets(msg):
    global PACKET_ORDER, PACKET_BUFFER, GLOBAL_MESSAGE_COUNTER
    PACKET_BUFFER.clear()
    packet_cut = FRAGMENT_SIZE - FRAGMENT_HEAD_SIZE
    cut_string = [msg[i:i + packet_cut] for i in range(0, len(msg), packet_cut)]

    for i in range(len(cut_string)):
        order = PACKET_ORDER
        PACKET_ORDER += 1
        order = order.to_bytes(4, "big")
        operation = OPERATION
        operation = operation.to_bytes(1, "big")
        total_packets = len(cut_string)
        total_packets = total_packets.to_bytes(4, "big")
        cut_msg = cut_string[i].encode()
        crc = libscrc.buypass(order + operation + total_packets + cut_msg)
        crc = crc.to_bytes(4, "big")
        fragment_msg = order + operation + total_packets + cut_msg + crc
        PACKET_BUFFER.append(copy(fragment_msg))
```

Príklad jednoduchšej komunikácie kde pošlem malú správu:



```
C:\Windows\System32\cmd.exe - python s.py
Microsoft Windows [Version 10.0.19045.2251]
(c) Microsoft Corporation. Všetky práva vyhradené.

C:\Users\Admin\PycharmProjects\pythonProject4>python s.py
Current host name is 192.168.56.1, select your new host (or type the same address to
192.168.56.1
Default port of this program is 5555, select your new port or keep the same port by t
5555
Choose fragment size:
1024
Connected to ('192.168.56.1', 58928)
b'\x00\x00\x00\x01\x03\x00\x00\x01hello, how are you?\x00\x0d\xf6'
Order: 1
Operation: WRT
Total_packets: 1
Message: hello, how are you?
Checksum: 40438
hello, how are you?

C:\Users\Admin\AppData\Local\Programs\Python\Python310\python.exe
Current host name is 192.168.56.1, select your new host (or type the same address to
192.168.56.1
Default port of this program is 5555, select your new port or keep the same port by t
5555
Choose fragment size:
1024
Are you sending a message or a file or switching roles?
(msg/file/switch)
msg
Enter message: hello, how are you?
[b'\x00\x00\x00\x01\x03\x00\x00\x01hello, how are you?\x00\x0d\xf6']
Packet number 0 was sent successfully. Yay :D
Are you sending a message or a file or switching roles?
(msg/file/switch)
```

Pre ukážku poslania väčšej správy pošlem preklad textu Lorem Ipsum z roku 1914 preloženým pánom H. Rackham:

"On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and demoralized by the charms of pleasure of the moment, so blinded by desire, that they cannot foresee the

pain and trouble that are bound to ensue; and equal blame belongs to those who fail in their duty through weakness of will, which is the same as saying through shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In a free hour, when our power of choice is untrammelled and when nothing prevents our being able to do what we like best, every pleasure is to be welcomed and every pain avoided. But in certain circumstances and owing to the claims of duty or the obligations of business it will frequently occur that pleasures have to be repudiated and annoyances accepted. The wise man therefore always holds in these matters to this principle of selection: he rejects pleasures to secure other greater pleasures, or else he endures pains to avoid worse pains."

Veľkosť fragmentu bude iba 200

```

C:\Windows\System32\cmd.exe - python s.py
Checksum: 15922
b'\x00\x00\x00\x05\x03\x00\x00\x06that pleasures have to be repudiated and annoyances accepted. The wise man therefore
Order: 5
Operation: _WRT
Total_packets: 6
Message: that pleasures have to be repudiated and annoyances accepted. The wise man therefore
ers to this principle of selection: he rejects pleasures to secure other gre
Checksum: 49470
b'\x00\x00\x00\x06\x03\x00\x00\x06ater pleasures, or else he endures pains to avoid worse
Order: 6
Operations: _WRT
Total_packets: 6
Message: ater pleasures, or else he endures pains to avoid worse pains."
Checksum: 16459

On the other hand, we denounce with righteous indignation and dislike men who are so beguiled
ymes of pleasure of the moment, so blinded by desire, that they cannot foresee the pain and tr
ue; and equal blame belongs to those who fail in their duty through weakness of will, which i
gh shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In
r of choice is untrammelled and when nothing prevents our being able to do what we like best,
icomed and every pain avoided. But in certain circumstances and owing to the claims of duty or
Packet number 0 was sent successfully. Yay :D
Packet number 1 was sent successfully. Yay :D
Packet number 2 was sent successfully. Yay :D
Packet number 3 was sent successfully. Yay :D
Packet number 4 was sent successfully. Yay :D
Packet number 5 was sent successfully. Yay :D
Are you sending a message or a file or switching roles?
(msg/file/switch)

```

Posielanie správ sa deje vo vnorenom while cykle:

```

if packettype == 'msg':
    OPERATION = WRT
    msg = input("Enter message: ")
    encode_multiple_packets(msg)
    for i in range(len(PACKET_BUFFER)):
        s.send(PACKET_BUFFER[i])

    ACK_packet = s.recv(FRAGMENT_SIZE)
    ACK_packet = decode_ACK(ACK_packet)

    while ACK_packet[1] == "_NCK":
        print(f"Failed to send packet number {i}, resending packet...\n")
        s.send(PACKET_BUFFER[i])
        ACK_packet = s.recv(FRAGMENT_SIZE)
        ACK_packet = decode_ACK(ACK_packet)

    print(f"Packet number {i} was sent successfully. Yay :D\n")

```

Prijímanie správ na strane servera:


```

while True:
    data = conn.recv(FRAGMENT_SIZE)
    print(data)
    if not data:
        break
    packet = decode_data(data)

```

Paket sa musí aj dekodovať než sa vypíše. To sa deje vo funkcii `decode_data()`.

```

def decode_data(data):
    global OPERATION
    operation = int.from_bytes(data[4:5], "big")
    opcode = get_flag(operation)
    if opcode == "_WRT":
        OPERATION = "_WRT"
        packet = decode_WRT(data, operation, opcode)
        return packet
    if opcode == "_END":
        OPERATION = "_END"
        packet = decode_END(data, operation, opcode)
        return packet
    if opcode == "_PFL":
        OPERATION = "_PFL"
        packet = decode_PFL(data, operation, opcode)
        return packet
    if opcode == "_KAR":
        OPERATION = "_KAR"
        packet = decode_KAR(data, operation, opcode)
        return packet
    if opcode == "_SWR":
        OPERATION = "_SWR"
        packet = decode_SWR(data, operation, opcode)
        return packet

```

Najprv musím zistiť, o aký typ paketu sa jedná. Nasledovne tento paket dekodujem.

```
def decode_WRT(data, operation, opcode):
    global SIMULATE_ERROR
    order = int.from_bytes(data[:4], "big")
    total_packets = int.from_bytes(data[5:9], "big")
    msg = data[9:-4]
    crc = int.from_bytes(data[-4:], "big")
    checksum = libscrc.buypass(order.to_bytes(4, "big") + operation.to_bytes(1, "big") +
                                total_packets.to_bytes(4, "big") + msg)

    if SIMULATE_ERROR:
        if random.randint(1, 3) == 1:
            checksum = checksum + checksum
            print("Wrong packet, requesting resending...")
    msg = msg.decode()
    if checksum == crc:
        packet = [order, opcode, total_packets, msg, crc]
        return packet
    else:
        packet = [0, "_NCK", 0, 0, 0]
        return packet
```

Počas tohto procesu zároveň aj kontrolujem checksum. Pokiaľ sa checksum rovná, tak viem že som dostal správny paket a môžem paket vypísať:

```
print(f"Order: {packet[0]}\n"
      f"Operation: {packet[1]}\n"
      f"Total_packets: {packet[2]}\n"
      f"Message: {packet[3]}\n"
      f"Checksum: {packet[4]}\n")
```

Toto ale vypíše iba samotný paket a nie celú správu. Celá správa sa vypíše, až keď som dostal všetky pakety danej komunikácie.

```
if packet[1] != "_NCK" and packet[1] != "_KAR":
    full_msg.append(copy(packet[3]))
    ACK_packet = encode_ACK()
    conn.send(ACK_packet)
```

```
if packet[0] == packet[2] and OPERATION == "_WRT" and packet[1] != "_NCK":
    str = list_to_string(full_msg)
    print(str)
    print("\n")
    del str
    full_msg.clear()
```

Keď sa mi bude rovnať poradie paketu s celkovým počtom paketov v jednej komunikácii tak viem, že som dostal celú správu tak ju aj vypíšem.

4.0 Posielanie súborov

Posielanie súborov je veľmi podobné tomu, ako posielam jednoduché správy. Používateľ ale musí najprv zadať, aký súbor chce poslať a v ktorom adresári sa nachádza. Súbor potom musím prečítať ako binárny súbor:

```
with open(filePath, "rb") as bin_file:
    fileContent = bin_file.read()
```

Vo fileContent mám bytes array súboru. Tento reťazec už iba musím rozložiť tak isto, ako som delil správy.

```
encode_file_packets(fileContent)
for i in range(len(PACKET_BUFFER)):
    print(PACKET_BUFFER[i])
    s.send(PACKET_BUFFER[i])

ACK_packet = s.recv(FRAGMENT_SIZE)
ACK_packet = decode_ACK(ACK_packet)
```

Na strane servera:

```
def decode_PFL(data, operation, opcode):
    order = int.from_bytes(data[:4], "big")
    total_packets = int.from_bytes(data[5:9], "big")
    msg = data[9:-4]
    crc = int.from_bytes(data[-4:], "big")
    checksum = libscrc.buypass(order.to_bytes(4, "big") + operation.to_bytes(1, "big") +
                                total_packets.to_bytes(4, "big") + msg)

    if SIMULATE_ERROR:
        if random.randint(1, 25) == 1:
            checksum = checksum + checksum
            print("Wrong packet")
    if checksum == crc:
        packet = [order, opcode, total_packets, msg, crc]
        return packet
    else:
        packet = [0, "_NCK", 0, 0, 0]
        return packet
```

Hlavný rozdiel medzi touto funkciou a funkciou decode_WRT() je ten, že v decode_PFL() nedekodujem dáta, ktoré mi prišli. Tie potrebujem, aby mi ostali ako binárny reťazec. Ten už iba spojím keď mi príde celá správa:

```
if packet[0] == packet[2] and OPERATION == "_PFL" and packet[1] != "_NCK":
    bytes_array_to_file(full_msg)
    full_msg.clear()
```

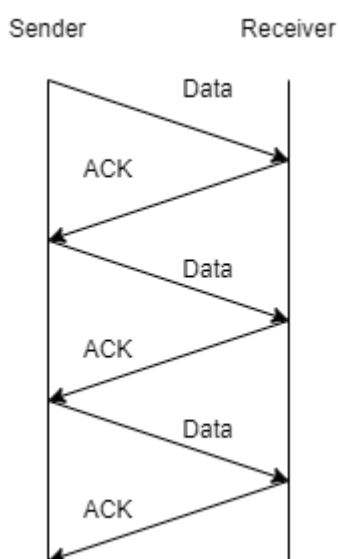
A na záver vo funkcii `bytes_array_to_file()` aj súbor uloží:

```
def bytes_array_to_file(data):
    bytes_array = bytearray()
    for i in data:
        bytes_array += i
    filePath = input("Enter download path: ")
    fileName = input("Enter name of the file: ")
    if os.path.exists(filePath):
        filePath = filePath + "\\ " + fileName
    with open(filePath, "wb+") as bin_file:
        bin_file.write(bytes_array)
    print(f"File successfully downloaded on {filePath}")
    return
```

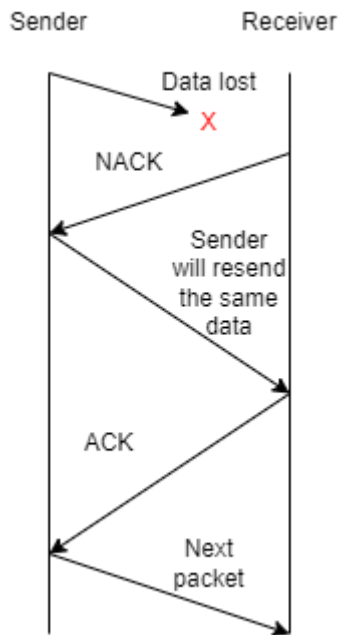
5.0 Implementácia ARQ metódy

V programe využívam **Stop & Wait** metódu. Po každom pakete klient bude čakať na ACK od servera. Pokiaľ ho nedostane alebo príde NCK, teda že server prijal zlý paket, tak ho klient pošle znova.

Príklad úspešnej komunikácie:



Príklad neúspešnej komunikácie:



Klient pošle paket a bude čakať na ACK od servera. Server vytvorí ACK paket vo funkcii:

```
def encode_ACK():
    order = 0
    opcode = 0
    msg = "Acknowledgement"
    total_packets = 1
    order = order.to_bytes(4, "big")
    opcode = opcode.to_bytes(1, "big")
    total_packets = total_packets.to_bytes(4, "big")
    msg = msg.encode()
    crc = libsrcrc.buypass(order + opcode + total_packets + msg)
    crc = crc.to_bytes(4, "big")
    packet = order + opcode + total_packets + msg + crc
    return packet
```

V prípade chyby ale server pošle NCK paket:

```
def encode_NCK():
    order = 0
    opcode = 6
    msg = "Not acknowledged"
    total_packets = 1
    order = order.to_bytes(4, "big")
    opcode = opcode.to_bytes(1, "big")
    total_packets = total_packets.to_bytes(4, "big")
    msg = msg.encode()
    crc = libscrc.buypass(order + opcode + total_packets + msg)
    crc = crc.to_bytes(4, "big")
    packet = order + opcode + total_packets + msg + crc
    return packet
```

Pokiaľ klientovi príde NCK, tak znova bude posielať stratený paket až dovtedy, pokiaľ mu server nepošle ACK:

```
while ACK_packet[1] == "_NCK":
    print(f"Failed to send packet number {i}, resending packet...\n")
    s.send(PACKET_BUFFER[i])
    ACK_packet = s.recv(FRAGMENT_SIZE)
    ACK_packet = decode_ACK(ACK_packet)
```

6.0 Simulovanie chyby a jej riešenie

Chybu simulujem pomocou náhody. To znamená, že keď mi funkcia `random.randint(1,25)` vráti číslo 1 tak vnesiem chybu do crc hashu. Vtedy viem, že mi prišiel zlý paket klientovi

pošlem NCK. Riešenie tejto chyby je opísané v časti 5.0.

```
def decode_PFL(data, operation, opcode):
    order = int.from_bytes(data[:4], "big")
    total_packets = int.from_bytes(data[5:9], "big")
    msg = data[9:-4]
    crc = int.from_bytes(data[-4:], "big")
    checksum = libscrc.buypass(order.to_bytes(4, "big") + operation.to_bytes(1, "big") +
                                total_packets.to_bytes(4, "big") + msg)

    if SIMULATE_ERROR:
        if random.randint(1, 25) == 1:
            checksum = checksum + checksum
            print("Wrong packet")

    if checksum == crc:
        packet = [order, opcode, total_packets, msg, crc]
        return packet
    else:
        packet = [0, "_NCK", 0, 0, 0]
        return packet
```

Keď mi príde chyba, tak vrátim NCK paket a tej potom server pošle späť klientovi.

7.0 Keep-alive metóda

Keep-alive metóda bola implementovaná pomocou vlákien.

```
### THREAD ###
keep_alive_thread = threading.Thread(target=send_keep_alive_request, args=(s,))
keep_alive_thread.start()
### THREAD ###
```

```
def send_keep_alive_request(s):
    while True:
        KAR_packet = encode_KAR()
        time.sleep(60)
        print("Sending Keep alive request")
        s.send(KAR_packet)
```

```
def encode_KAR():
    order = 0
    operation = KAR
    total_packets = 1
    msg = "Keep alive request"
    order = order.to_bytes(4, "big")
    operation = operation.to_bytes(1, "big")
    total_packets = total_packets.to_bytes(4, "big")
    msg = msg.encode()
    crc = libscrc.buypass(order + operation + total_packets + msg)
    crc = crc.to_bytes(4, "big")
    packet = order + operation + total_packets + msg + crc
    return packet
```

Táto metóda vytvorí a pošle KAR paket každých 60 sekúnd. Server tento paket prijme a pošle späť pokiaľ ho nedostane, tak vieme, že komunikácia sa ukončila.

8.0 Ukážka mojej komunikácie vo Wireshark

Komunikoval som so svojim počítačom cez Wifi. IP môjho počítača je 192.168.100.19 a IP môjho notebook-u je 192.168.100.16 na porte 5555. Pokúsil som sa preniesť obrázok z môjho počítača na môj notebook

Príklad komunikácie medzi nimi:

ip.addr == 192.168.100.16							
No.	Time	Source	Destination	Protocol	Length	Info	
3426	20.512193	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41273 Ack=1510401 Win=509 Len=28
3427	20.514880	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1510401 Ack=41301 Win=1025 Len=1024
3428	20.515585	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41301 Ack=1511425 Win=513 Len=28
3429	20.518108	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1511425 Ack=41329 Win=1025 Len=1024
3430	20.518802	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41329 Ack=1512449 Win=509 Len=28
3431	20.521360	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1512449 Ack=41357 Win=1025 Len=1024
3432	20.521991	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41357 Ack=1513473 Win=513 Len=28
3433	20.524574	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1513473 Ack=41385 Win=1025 Len=1024
3434	20.525208	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41385 Ack=1514497 Win=509 Len=28
3435	20.529275	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1514497 Ack=41413 Win=1025 Len=1024
3436	20.530049	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41413 Ack=1515521 Win=513 Len=28
3437	20.532407	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1515521 Ack=41441 Win=1025 Len=1024
3438	20.533121	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41441 Ack=1516545 Win=509 Len=28
3439	20.536046	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1516545 Ack=41469 Win=1025 Len=1024
3440	20.536675	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41469 Ack=1517569 Win=513 Len=28
3441	20.540253	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1517569 Ack=41497 Win=1025 Len=1024
3442	20.540883	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41497 Ack=1518593 Win=509 Len=28
3444	20.543464	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1518593 Ack=41525 Win=1024 Len=1024
3445	20.544117	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41525 Ack=1519617 Win=513 Len=28
3446	20.549112	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1519617 Ack=41553 Win=1024 Len=1024
3447	20.549730	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41553 Ack=1520641 Win=509 Len=28
3448	20.553181	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1520641 Ack=41581 Win=1024 Len=1024
3449	20.553779	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41581 Ack=1521665 Win=513 Len=28
3450	20.556401	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1521665 Ack=41609 Win=1024 Len=1024
3451	20.557004	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41609 Ack=1522689 Win=509 Len=28
3452	20.559641	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1522689 Ack=41637 Win=1024 Len=1024
3453	20.560292	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41637 Ack=1523713 Win=513 Len=28
3454	20.564210	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1523713 Ack=41665 Win=1024 Len=1024
3455	20.564819	192.168.100.19	192.168.100.16	TCP	82	5555 → 56789	[PSH, ACK] Seq=41665 Ack=1524737 Win=509 Len=28
3456	20.567146	192.168.100.16	192.168.100.19	TCP	1078	56789 → 5555	[PSH, ACK] Seq=1524737 Ack=41693 Win=1024 Len=1024

Príklady paketov:

Môj ACK paket:


```

Wireshark · Packet 3465 · Wi-Fi
> Frame 3465: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface \Device\NPF_{7CCE838D-3F2A-411F-8ED6-12610C4904DA}, id 0
> Ethernet II, Src: AzureWav_02:9d:0d (00:e9:3a:02:9d:0d), Dst: ASUSTekC_1d:99:84 (f0:2f:74:1d:99:84)
> Internet Protocol Version 4, Src: 192.168.100.19, Dst: 192.168.100.16
> Transmission Control Protocol, Src Port: 5555, Dst Port: 56789, Seq: 41805, Ack: 1529857, Len: 28
▼ Data (28 bytes)
  Data: 0000000000000000141636b6e6f776c656467656d656e740000766c
  [Length: 28]

0000  f0 2f 74 1d 99 84 00 e9 3a 02 9d 0d 08 00 45 00  ./t-----:-----E-
0010  00 44 59 fd 40 00 80 06 57 42 c0 a8 64 13 c0 a8  .DY.@...WB..d...
0020  64 10 15 b3 dd d5 7a aa 88 3f 40 5b fa c4 50 18  d-----z-?@[...P-
0030  02 01 c9 0f 00 00 00 00 00 00 00 00 00 00 01 41  .....:.....A
0040  63 6b 6e 6f 77 6c 65 64 67 65 6d 65 6e 74 00 00  cknowledgement..
0050  76 6c

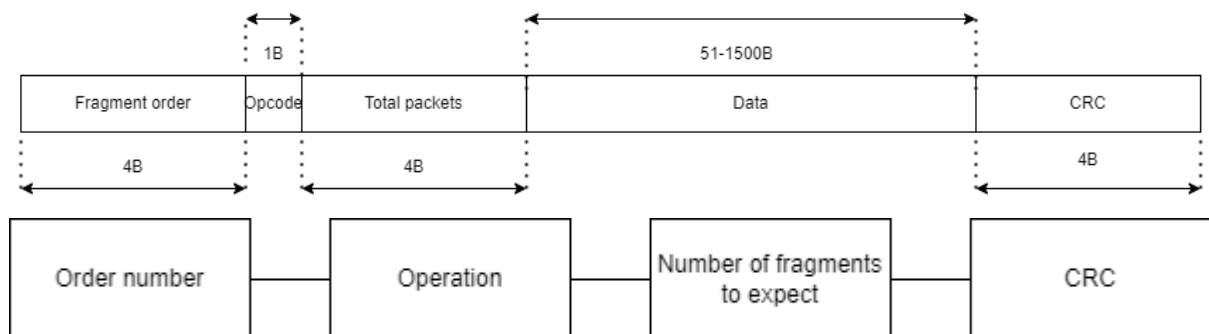
```

9.0 Zmeny medzi finálnou verzou a návrhom

1. Rozdielne hlavičky:

Kvalita a presnosť hlavičky je oveľa vyššia vo finálnej verzii.

Nová hlavička versus stará:



V novej hlavičke sú uvedené presné bity aj je sformovaná lepšie podľa pokynov predmetu.

2. Iná ARQ metóda

V návrhu som uviedol, že budem robiť **Selective Repeat** ale vo finálnom odovzdaní som použil **Stop & Wait** protokol.

3. Odlišné operácie, alebo iný opcode.

V návrhu som uviedol operácie ktoré som vôbec nevyužil vo finálnej verzii alebo som vo finálnej verzii pridal nové operácie. Napr. SYN som vôbec nepoužil ako som mal v návrhu a vo finálnom riešení som mal nové operácie PFL a SWR ktoré neboli v mojom návrhu.

10.0 Záver

Program by mal dostatočne dobre posielat' správy a súbory medzi dvoma uzlami. Program bol vypracovaný s pomocou prednášok pána prof. Ing. Ivana Kotuliaka, PhD a videí od Neso Academy ktorý mi lepšie objasnil fungovanie ARQ metódy (<https://www.youtube.com/@nesoacademy>).

Program celkovo zaberá dva súbory, s.py a c.py, a každý z nich má okolo 500 riadkov.