

FIIT STU

PKS Zadanie 2 Dokumentácia

Komunikátor

Meno: Jakub Gašparín

Cvičenie: Piatok 8:00-9:40

Cvičiacci: Ing. Miroslav Bahleda, PhD.

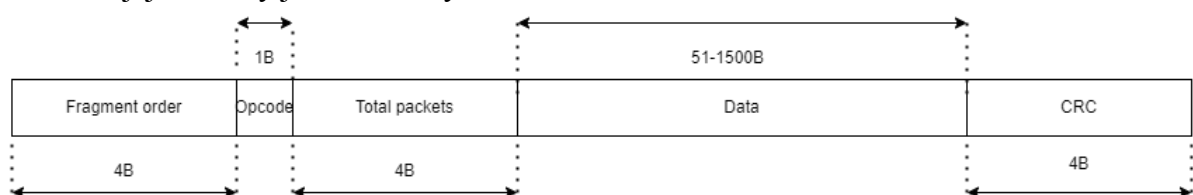
Obsah

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	Subor	Datum uprav	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 INITIAL_ACK, HOST, PORT, TOTAL_SENT, TOTAL_RECEIVED
    INITIAL_ACK = False
    HOST = "192.168.56.1"
    PORT = 5555
    HOST = input(f"Current host IP is {HOST}, please select your server IP: ")
    PORT = int(input(f"Current port is {PORT}, please select your port: "))
    s = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    s.bind((HOST, PORT))
    print("UDP server up and listening")
```

Klient sa spojí so serverom nasledovne:

```
def client():
    global INITIAL_ACK, HOST, PORT, OPERATION, PACKET_ORDER, TOTAL_SENT, TOTAL_RECEIVED
    INITIAL_ACK = False
    HOST = "192.168.56.1"
    PORT = 5555
    HOST = input(f"Current host IP is {HOST}, please select your server IP: ")
    PORT = int(input(f"Current port is {PORT}, please select your port: "))
    serverAddressPort = (HOST, PORT)
    c = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
```

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:

```
if packetType == "msg":
    msg = input("type your message: ")

    OPERATION = WRT
    encode_multiple_packets(msg)
    for i in range(len(PACKET_BUFFER)):
        c.sendto(PACKET_BUFFER[i], serverAddressPort)
        ACK_packet = c.recvfrom(FRAGMENT_SIZE)
        ACK_packet = ACK_packet[0]
        ACK_packet = decode_ACK(ACK_packet)
```

2.1: Prepnutie rolí servera a klienta

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

```
elif packetType == "switch":  
    SWR_packet = encode_SWR()  
    c.sendto(SWR_packet, serverAddressPort)  
    break
```

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

3.0 Posielanie jednoduchých správ a fragmentácia

Ako prvú vec čo som spravil bolo zabezpečenie jednoduchej 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"))
```

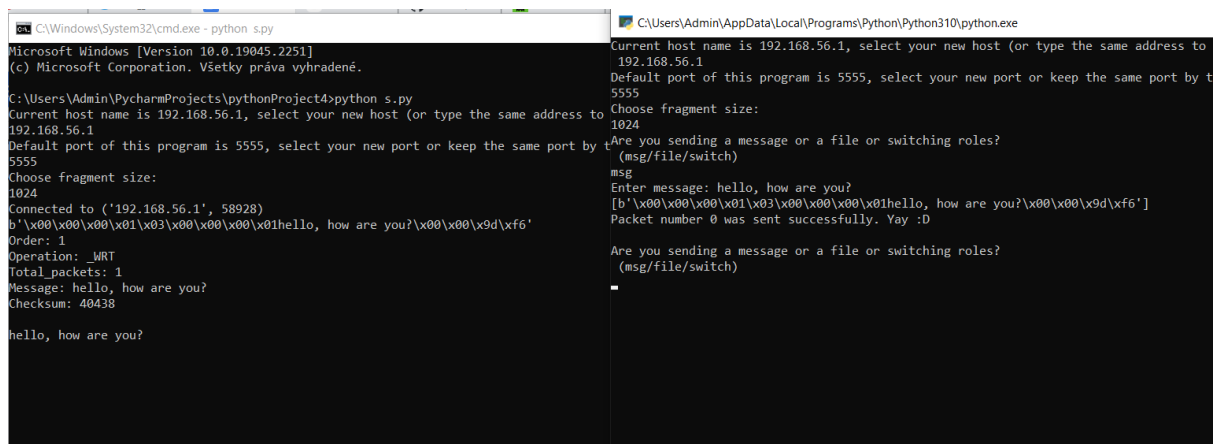
Na toto sa program vždy spýta pred začiatkom každej komunikácie.

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
5555
Choose fragment size:
1024
Connected to ('192.168.56.1', 58928)
b'\x00\x00\x00\x01\x03\x00\x00\x00\x01hello, how are you?\x00\x00\x9d\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
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\x00\x01hello, how are you?\x00\x00\x9d\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\x00\x06' 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
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\x05\x03\x00\x00\x00\x06' ater pleasures, or else he endures pains to avoid worse
Order: 6
Operation: _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
arms 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
sh shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In
s of choice is untrammelled and when nothing prevents our being able to do what we like best,
lcomed and every pain avoided. But in certain circumstances and owing to the claims of duty or
ss it will frequently occur that pleasures have to be repudiated and annoyances accepted. The
holds in these matters to this principle of selection: he rejects pleasures to secure other g
e endures pains to avoid worse pains."

ure is to be welcomed and every pain avoided. But in certain circumstances and owing to the claims of duty
tions of business it will frequently occur that pleasures have to be repudiated and annoyances accepted. Th
erefore always holds in these matters to this principle of selection: he rejects pleasures to secure other
ures, or else he endures pains to avoid worse pains."
[b'\x00\x00\x00\x01\x03\x00\x00\x00\x06' "On the other hand, we denounce with righteous indignation and dislik
e so beguiled and demoralized by the charms of pleasure of the moment, so blinded by desire, that they can\
b'\x00\x00\x00\x02\x03\x00\x00\x00\x06' not foresee the pain and trouble that are bound to ensue; and equal
s to those who fail in their duty through weakness of will, which is the same as saying through shrinking f
xef', b'\x00\x00\x00\x03\x03\x00\x00\x00\x06' from toil and pain. These cases are perfectly simple and easy to
In a free hour, when our power of choice is untrammelled and when nothing prevents our being able to do wh
\x00', b'\x00\x00\x00\x04\x03\x00\x00\x00\x06' like best, every pleasure is to be welcomed and every pain a
n certain circumstances and owing to the claims of duty or the obligations of business it will frequently o
22', b'\x00\x00\x00\x05\x03\x00\x00\x00\x06' that pleasures have to be repudiated and annoyances accepted. Th
erefore always holds in these matters to this principle of selection: he rejects pleasures to secure other
c1>', b'\x00\x00\x00\x06\x03\x00\x00\x00\x06' ater pleasures, or else he endures pains to avoid worse pains."
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":
    msg = input("type your message: ")

    OPERATION = WRT
    encode_multiple_packets(msg)
    for i in range(len(PACKET_BUFFER)):
        c.sendto(PACKET_BUFFER[i], serverAddressPort)
        ACK_packet = c.recvfrom(FRAGMENT_SIZE)
        ACK_packet = ACK_packet[0]
        ACK_packet = decode_ACK(ACK_packet)
        while ACK_packet[1] == "_NCK":
            print(f"Failed to send packet number {i+1}, resending packet...\n")
            c.sendto(PACKET_BUFFER[i], serverAddressPort)
            ACK_packet = c.recvfrom(FRAGMENT_SIZE)
            ACK_packet = ACK_packet[0]
            ACK_packet = decode_ACK(ACK_packet)

        print(f"Packet number {i+1} was sent successfully. Yay :D\n")
```

Prijímanie správ na strane servera:

```
msg = s.recvfrom(FRAGMENT_SIZE)
msg = msg[0]
packet = decode_data(msg)
```

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()
    s.sendto(ACK_packet, address)
```

```
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.

```
if os.path.exists(filePath):
    filePath = filePath + "\\\" + fileName
    if os.path.exists(filePath):
        with open(filePath, "rb") as bin_file:
            fileContent = bin_file.read()
            encode_file_packets(fileContent)
            for i in range(len(PACKET_BUFFER)):
                c.sendto(PACKET_BUFFER[i], serverAddressPort)

                ACK_packet = c.recvfrom(FRAGMENT_SIZE)
                ACK_packet = ACK_packet[0]
                ACK_packet = decode_ACK(ACK_packet)

                while ACK_packet[1] == "_NCK":
                    print(f"Failed to send packet number {i+1}, resending packet..")
                    c.sendto(PACKET_BUFFER[i], serverAddressPort)
                    ACK_packet = c.recvfrom(FRAGMENT_SIZE)
                    ACK_packet = ACK_packet[0]
                    ACK_packet = decode_ACK(ACK_packet)

                print(f"Packet number {i+1} was sent successfully. Yay :D\\n")
```

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:

```

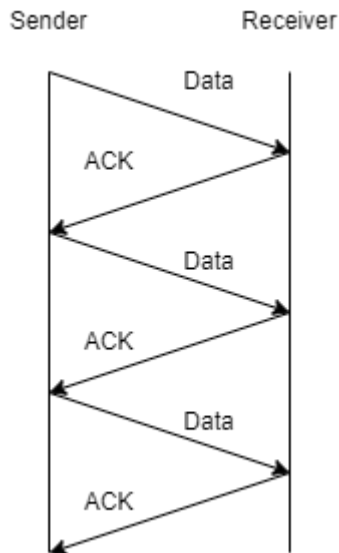
def bytes_array_to_file(data):
    bytes_array = bytearray()
    for i in data:
        bytes_array += i
    filePath = input("Enter download path: ")
    fileName = input("Enter name of the file: ")
    if os.path.exists(filePath):
        filePath = filePath + "\\ " + fileName
    with open(filePath,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

```

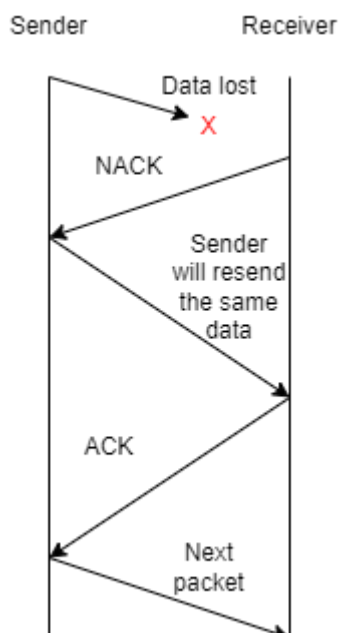
5.0 Implementácia ARQ metódy

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 = libscrc.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 posielat' stratený paket až dovtedy, pokiaľ mu server nepošle ACK:

```

while ACK_packet[1] == "_NCK":
    print(f"Failed to send packet number {i+1}, resending packet...\n")
    c.sendto(PACKET_BUFFER[i], serverAddressPort)
    ACK_packet = c.recvfrom(FRAGMENT_SIZE)
    ACK_packet = ACK_packet[0]
    ACK_packet = decode_ACK(ACK_packet)

print(f"Packet number {i+1} was sent successfully. Yay :D\n")

```

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(c, addr):
    while True:
        KAR_packet = encode_KAR()
        time.sleep(10)
        # print("Sending Keep alive request")
        c.sendto(KAR_packet, addr)
```

```
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:

8412	24.118276	192.168.100.19	192.168.100.8	UDP	58 52790 → 5555	Len=16
8413	24.124402	192.168.100.8	192.168.100.19	UDP	58 5555 → 52790	Len=16
8465	28.240011	192.168.100.19	192.168.100.8	UDP	57 52790 → 5555	Len=15
8466	28.247865	192.168.100.8	192.168.100.19	UDP	70 5555 → 52790	Len=28
10479	34.132210	192.168.100.19	192.168.100.8	UDP	73 52790 → 5555	Len=31
14497	44.137348	192.168.100.19	192.168.100.8	UDP	73 52790 → 5555	Len=31
17015	48.308672	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200
17016	48.316767	192.168.100.8	192.168.100.19	UDP	70 5555 → 52790	Len=28
17017	48.317622	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200
17018	48.322863	192.168.100.8	192.168.100.19	UDP	70 5555 → 52790	Len=28
17019	48.323124	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200
17020	48.328739	192.168.100.8	192.168.100.19	UDP	70 5555 → 52790	Len=28
17021	48.329075	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200
17022	48.335534	192.168.100.8	192.168.100.19	UDP	70 5555 → 52790	Len=28
17023	48.336059	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200
17024	48.340834	192.168.100.8	192.168.100.19	UDP	70 5555 → 52790	Len=28
17025	48.340993	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200
17026	48.346607	192.168.100.8	192.168.100.19	UDP	70 5555 → 52790	Len=28
17027	48.347256	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200
17028	48.352717	192.168.100.8	192.168.100.19	UDP	70 5555 → 52790	Len=28
17029	48.353667	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200
17030	48.359841	192.168.100.8	192.168.100.19	UDP	71 5555 → 52790	Len=29
17031	48.362068	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200
17032	48.368406	192.168.100.8	192.168.100.19	UDP	70 5555 → 52790	Len=28
17033	48.369355	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200
17034	48.375667	192.168.100.8	192.168.100.19	UDP	70 5555 → 52790	Len=28
17035	48.377928	192.168.100.19	192.168.100.8	UDP	242 52790 → 5555	Len=200

<

> Frame 8465: 57 bytes on wire (456 bits), 57 bytes captured (456 bits) on interface \Device\NPF_{7CCE838D-3F2A-411F-8ED6-12610C4904D} Ethernet II, Src: AzureWav_02:9d:0d (00:e9:3a:02:9d:0d), Dst: Chongqin_11:c9:53 (b0:68:e6:11:c9:53)

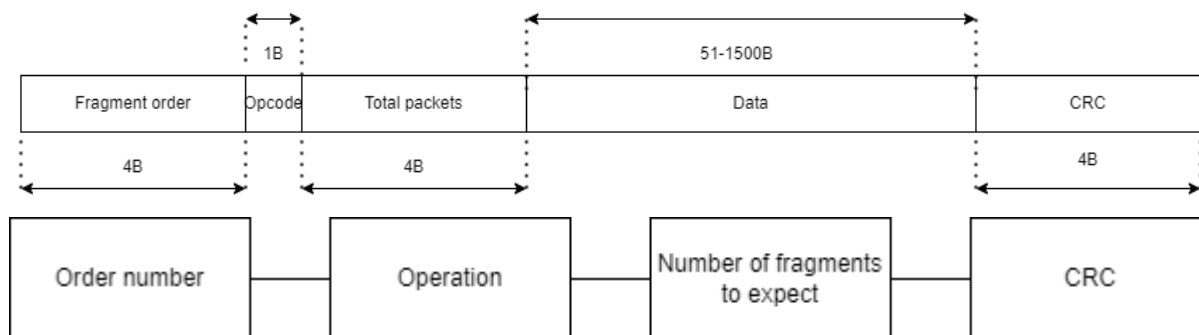
0000	b0 68 e6 11 c9 53 00 e9 3a 02 9d 0d 08 00 45 00	..h...S...E..
0010	00 2b 65 57 00 00 80 11 8b fe c0 a8 64 13 c0 a8	...+eW....d...
0020	64 08 ce 36 15 b3 00 17 13 1e 00 00 00 01 03 00	d..6.....
0030	00 00 01 68 69 00 00 e2 51	...hi... Q

9.0 Zmeny medzi finálnou verziou 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 sfomovaná 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.