Telekomunikacja - laboratorium				Studia stacjonarne - inżynierskie	
Nazwa zadania	**Ćwiczenie nr 2. Protokół Xmodem**				
Dzień	**poniedziałek**	**Godzina**	**14:00-15:30**	**Rok akademicki**	**2020/2021**

Imię i Nazwisko	**Adam Kapuściński 229907**
Imię i Nazwisko	**Damian Szczeciński 230016**
Imię i Nazwisko	**-**

Opis programu, rozwiązania problemu.

Celem zadania było napisanie programu umożliwiającego wysyłanie jak i odbieranie przesyłanych danych wykorzystując do tego porty szeregowy RS-232 (w naszym przypadku wirtualne). Program miał w założeniach korzystać z technologii Xmodem z obsługą sumy kontrolnej liczonej jako suma algebraiczna, oraz algorytmu CRC.

Zasada działania Xmodem:

Xmodem jest protokołem komunikacji półdupleksowej. Komunikację w jego przypadku jest inicjowana przez odbiornik wysyłający znak NAK. Nadajnik w tym momencie definiuje jakim sposobem będzie obliczana suma kontrolna (C <- CRC, NAK <- suma algebraiczna). Po czym rozpoczyna się przesyłanie pakiety po pakiecie. Po odebraniu pakiety, jeżeli zgadza się suma kontrolna, odsyłany jest znak ACK, po którym przesyłany jest kolejny blok informacji. Jeżeli suma kontrolna się nie zgadza odbiornik wysyła znak NAK i nadajnik ponawia transmisję błędnie przesłanego bloku danych. Po otrzymaniu potwierdzenia poprawnie przesłanego ostatniego bloku nadajnik przesyła znak EOT, oznaczający koniec, na który musi dostać odpowiedź ACK aby poprawnie zamknąć połączenie.

Format pakietu Xmodem:

- 1B -> początek nagłówka (SOH)
- 2B -> numer pakietu
- 3B -> numer pakietu
- 4B-131B -> Dane pakietu
- 132B-133B -> 16bitowa suma kontrolna CRC / 8bitowa suma algebraiczna

Numer pakietu powinien wyglądać w ten sposób, aby dodając te dwa bajty razem powinny być zawsze równe 0xFF.

Najważniejsze elementy kodu programu z opisem.

Funkcja wysyłająca

Poniższy kod ma na celu wysłanie wybranego przez użytkownika zbioru informacji. Umożliwiony w nim jest wybór numeru portu szeregowego.

```
1.  file = FileLoader(input("Podaj ścieżkę do pliku do wysłania: "))
2.
3.  # wylistowanie dostępnych portów
4.  ports = sorted(lp.comports())
5.  res_ports = []
6.  print("Dostępne porty:")
7.  i=0
8.  for port, desc, hwid in ports:
9.      res_ports.append(port)
10.     print(str(i) + ". " + port)
11.     i += 1
12. if len(res_ports) == 0: exit("Brak dostępnych portów")
13. choosen_port = res_ports[int(input("Wybierz: "))]
14.
15. # otwarcie połączenia
16. print("Otwieram połączenie na " + choosen_port)
17. conn = ps.Serial(choosen_port, timeout=3, inter_byte_timeout=1) # inter_byte_timeout=1 <- odpowiada za
    poprawne działanie, nowe komputery są za szybkie
18. conn.flush()
19.
20.
21. print("Oczekuję na rozpoczęcie transmisji")
22. while True:
23.     method = conn.read(1)
24.     if method == b'': continue          # nic nieodebrano = czekaj dalej
25.     if method == NAK: method = ch.sum   # NAK = suma kontrolna to suma algebraiczna
26.     if method == C  : method = ch.crc   # c   = suma kontrolna to CRC
27.     break
28.
29. print("Wykryto chęć odebrania pliku. Rozpoczynam transmisję")
30.
31. packet = XPacketSender(method)
32.
```

```
while not file.isEOF():
    conn.write(packet.sendPacket(file.readNext()))
    while True:
        tmp = conn.read(1)
        # Jeżeli ACK to przejdź do kroku dalej
        if tmp == ACK: break
        if tmp == b'': continue
        # Jeżeli NAK to powtórz
        if tmp == NAK:
            conn.write(packet.sendPacket())
            continue

print("Wysłałem plik. Kończę transmisję")

while True:
    # Jak koniec to wyślij EOT
    conn.write(EOT)
    tmp = conn.read(1)
    # Jeżeli nie otrzymano ACK (potwierdzenie odebrania wiadomości o końcu) powtórz
    if tmp == ACK: break

print("Transmisja zakończona")
```

Funkcja odbierania

Poniższa funkcja ma na celu odbieranie przesyłanego zbioru informacji. Również można tutaj wybrać port, ale wyróżnikiem jest możliwość wyboru sposobu obliczania sumy kontrolnej zgodnie z założeniami zadania i metody Xmodem.

```
 # gdzie zapisać plik
outFile = open(input("Podaj jak zapisać odbierany plik: "), 'wb')

# wybranie rodzaju sumy kontrolnej
control_sum = None
bytes_to_read = 132 # default jako suma kontrolna
print("1. Suma algebraiczna")
print("2. CRC")
print("Cokolwiek. Wyjscie z programu")
choice = input("Wybierz sposób potwierdzenia transmisji: ")
if choice == "1":
    control_sum = ch.sum
    C = NAK
elif choice == "2":
    control_sum = ch.crc
    bytes_to_read += 1 # CRC potrzebuje dodatkowego bajtu
else: exit()

# wylistowanie dostępnych portów
ports = sorted(lp.comports())
res_ports = []
print("Dostępne porty:")
i=0
for port, desc, hwid in ports:
    res_ports.append(port)
    print(str(i) + ". " + port)
    i += 1
if len(res_ports) == 0: exit("Brak dostępnych portów")
choosen_port = res_ports[int(input("Wybierz: "))]

print("Otwieram połączenie na " + choosen_port)
conn = ps.Serial(choosen_port, timeout=3, inter_byte_timeout=1) # inter_byte_timeout=1 <- odpowiada za poprawne działanie, nowe komputery są za szybkie
conn.flush()

# zainicjuj połączenie
buffer = XPacketReceiver()
while len(buffer.getData()) == 0:
    # wysłanie C <- zainicjiowanie odbioru z sumą kontrolną CRC
    conn.write(C)
    tmp = conn.read(bytes_to_read)
    if len(tmp) > 0: buffer.newPacket(tmp)

print("Połączenie ustanowione. Odbieram dane")

while buffer.getStart() != EOT:
    if not buffer.checkChecksum(control_sum):
        # wysłanie znaku sterującego NAK, prośba ponownego wysłania
        conn.write(NAK)
        # ponowne odczytanie pakietu
        tmp = conn.read(bytes_to_read)
```

```
51.         # jeżeli odebrany pakiet ma właściwą długość jest przekazywany dalej
52.         if len(tmp) == bytes_to_read: buffer.newPacket(tmp)
53.         continue
54.     # zapis widomości do pliku
55.     outFile.write(buffer.getData())
56.     # czyszczenie buforu
57.     conn.flush()
58.     # Wysłanie ACK <- potwierdzenie poprawnego odebrania pakietu
59.     conn.write(ACK)
60.
61.     i = 0
62.     for i in range(20):
63.         # zapis otrzymanego pakietu
64.         tmp = conn.read(bytes_to_read)
65.         if len(tmp) > 0:
66.             # jeżeli długość jest większa od 0 to przekazuje otrzymany pakiet dalej
67.             buffer.newPacket(tmp)
68.             break
69.     # jeżeli przez minutę nie uda się otrzymać wiadomości, to zwróć błąd
70.     if i == 19:
71.         conn.write(CAN)
72.         exit("Timeout")
73.
74. # zakończenie połączenia (otrzymano EOT)
75. conn.write(ACK)
76.
77. print("Połączenie zakończone. Plik odebrany")
78.
```

Funkcja wyliczająca sumy kontrolne

Poniżej zamieszczamy kod odpowiedzialny za wyliczanie sum kontrolnych algebraicznie i CRC.

```
1.   def crc(data : bytes):
2.       # funkcja generująca nagłówek CRC
3.       # zmiana na tablicę bajtową
4.       data = bytearray(data)
5.       crc = 0
6.       for byte in data:
7.           crc ^= byte << 8
8.           for i in range(8):
9.               if crc & 0x8000:
10.                  crc = (crc << 1) ^ 0x1021
11.              else:
12.                  crc = crc << 1
13.      return (crc & 0xFFFF).to_bytes(2, "big")
14.
15. def sum(data : bytes):
16.      # funkcja generująca sumę algebraiczną
17.      data = bytearray(data)
18.      result = 0
19.      for byte in data:
20.          result += byte
21.          result %= 256
22.      return result.to_bytes(1, "big")
23.
24.
```

Klasa odpowiedzialna za wysyłanie

Jest to fragment kodu w którym zamieszczone zostały definicje odpowiedzialne za wysyłanie.

```
1.   class XPacketSender:
2.
3.       method                  = None
4.       start                   = SOH
5.       packet_number           = 0
6.       packet number reversed  = 255
7.       data                    = b''
8.       checksum                = b''
9.
10.      def __init__(self, method):
11.          # zapisanie metody obliczania sumy kontrolnej
12.          self.method = method
13.
```

```
14.     def sendPacket(self, data=None):
15.         # jeżeli podano dane, to zaaktualizuj klasę
16.         # jeżeli nie, to wyślij to samo co wcześniej
17.         if data != None:
18.             self.__iteratePacketNumber()
19.             self.data = data
20.             self.checksum = self.method(self.data)
21.
22.         # połącz dane i wyślij
23.         return self.start\
24.             + self.packet_number.to_bytes(1, "big")\
25.             + self.packet_number_reversed.to_bytes(1, "big")\
26.             + self.data\
27.             + self.checksum
28.
29.     def __iteratePacketNumber(self):
30.         # ziteruj numer pakietu
31.         self.packet_number          += 1
32.         self.packet_number_reversed -= 1
33.         # jeżeli przekracza zakres, to napraw
34.         if self.packet_number > 255:
35.             self.packet_number = 0
36.         if self.packet_number_reversed < 0:
37.             self.packet_number_reversed = 255
38.
```

Klasa odpowiedzialna za odbieranie

Jest to fragment kodu w którym zamieszczone zostały definicje odpowiedzialne za odbieranie.

```
1.   class XPacketReceiver:
2.
3.       start                   = SOH
4.       packet_number           = 0
5.       packet_number_reversed  = 255
6.       data                    = b''
7.       checksum                = b''
8.
9.       def newPacket(self, buffer):
10.          # pierwszy Bajt - nagłówek SOH
11.          self.start                  = buffer[0].to_bytes(1, "big")
12.          if len(buffer) == 1: return
13.          # drugi Bajt - numer pakietu
14.          self.packet_number          = buffer[1]
15.          # trzeci Bajt - 255 odjąć numer pakietu
16.          self.packet_number_reversed = buffer[2]
17.          # wyczyszczenie starych danych
18.          self.data                   = b''
19.          # bajty 4-131 - dane
20.          for i in range(3, 131):
21.              self.data += buffer[i].to_bytes(1, "big")
22.          # bajty 132-133 - suma kontrolna
23.          self.checksum = buffer[131].to_bytes(1, "big")
24.          # jeżeli jest to CRC to dodajemy jeszcze jeden Bajt
25.          if len(buffer) == 133:
26.              self.checksum += buffer[132].to_bytes(1, "big")
27.
28.      def getStart(self):
29.          return self.start
30.
31.      def getPacketNumber(self):
32.          return self.packet_number
33.
34.      def getPacketNumberReversed(self):
35.          return self.packet_number_reversed
36.
37.      def getData(self):
38.          return self.data
39.
40.      def getChecksum(self):
41.          return self.checksum
42.
43.      def checkChecksum(self, method):
44.          # przeliczenie sumy kontrolnej danych i porównanie
45.          # z sumą otrzymaną w pakiecie
46.          # metodą podaną przez użytkownika (suma algb lub crc)
```

```
47. return method(self.data) == self.checksum
```

Podsumowanie wnioski.

Załączony do sprawozdania program może bez problemu komunikować się z innymi programami obsługującymi protokół Xmodem, więc możemy stwierdzić, że zadanie zostało wykonane poprawnie. Po poddaniu protokołu testom jesteśmy w stanie przyznać, że nie sprawdziłby się on w obecnych czasach. Powodem takiego stanu rzeczy jest jego ograniczenia w kontekście prędkości przesyłu danych. Wg naszych testów i wyliczeń plik ważący ~225kB jest transmitowany przez nieco ponad 3 minuty. Wynik taki jest na dzisiejsze standardy nieakceptowalny. Pomimo jego mankamentów jesteśmy w stanie powiedzieć, że dzięki prostocie budowy tego protokołu jest on idealnym wyborem do nauki zasad działania komunikacji w sieci a także problemów, jakie trzeba uwzględnić przy projektowaniu programu obsługującego dany protokół.