SOLT Seri port protokolü - Veri alma ve kod çözme giriş

Geliştiriciler sıklıkla SOLT cihazlarının entegrasyonunu ve bir API sağlanmasını talep ederler, ancak aynı zamanda deneyimsizlik nedeniyle seri port ile çalışmaktan kaçınırlar. Oysa aslında, COM (Seri) port ile çalışmak oldukça basit ve sezgiseldir. Bazı durumlarda, programda kendi veri okuma işleminizi uygulamak, API'li bir ara sunucu kurmaktan ve sürdürmekten çok daha kullanışlı ve verimlidir.

Hadi bunu nasıl yapacağımızı bulalım - sadece 5-10 dakika sürecek!

Düğme veya çağrı paneli bir radyo vericisidir ve SOLT SR5-MPRT cihazı alıcı - modem olarak kullanılır. SR5 -MPRT modem, dahili USB dönüştürücü ve USB A bağlantı arayüzüne sahip ve DB9 bağlantı arayüzüne sahip olmak üzere iki versiyonu bulunan RS232 arayüzlü bir cihazdır. Esasen, bunlar aynı protokollere sahip özdeş cihazlardır.

Her cihazda (uzaktan kumanda, SOLT düğmesi) fabrikada programlanmış benzersiz bir kod bulunur; bu kod, sinyalin kaynağını benzersiz bir şekilde tanımlamanıza olanak tanır. Bu koda remote_id denir ve bir veri paketinde iletilir (Örnek 41 31 42 32 43 33 ("A1B2C3")).

Benzersiz cihaz tanımlayıcısının yanı sıra, her uzaktan kumanda bir veya daha fazla düğme içerir ve bu düğmelere basıldığında ilgili kod iletilir. Düğme kodları çeşitli eylemleri gösterir:

Çağrı Düğmeleri	**Hex**	**İptal Düğmeleri**	**Hex**
1	0x14	2	0x24
3	0x28	4	0x34
5	0x38	6	0x44
7	0x48	8	0x54
9	0x58	10	0x64
11	0x68	12	0x74
13	0x78	15-İptal	0xE4
		16- Tüm aramaların tamamen iptali	0xF4

Örneğin, SB9-3XBK konsolunda 1 = 14 ve 3 = 28 olmak üzere 2 arama düğmesi ve 15 = E4 olmak üzere bir iptal düğmesi bulunur.

SB7-1PBK konsolunda yalnızca bir çağrı düğmesi vardır 1 = 14

SOLT protokolünün açıklaması

SOLT protokolünde veri iletimi, 19 baytlık sabit uzunluktaki paketler halinde gerçekleştirilir. Paket formatı aşağıdaki gibidir:

Bayt	Amaç	Tanım
1	Başlangıç sembolü (STX)	Paketin başlangıcını tanımlar.
2-3	Basılan düğme kodu	Düğmeyi tanımlayan iki ASCII karakteri
4	Ayırıcı	Sabit ayırıcı
5-10	Uzaktan Kimlik	Benzersiz cihaz tanımlayıcısı (ASCII)
11	Ayırıcı	Sabit ayırıcı
12-18	Kullanıcı verileri	Kullanıcı kimliği içeren ASCII dizesi
19	Bitiş sembolü (ETX)	Paket tamamlandı

Onaltılık sistemde bir paket örneği:

02 31 34 2D 41 31 42 32 43 33 2D 31 32 33 34 35 36 37 03

Nerede:

02 (STX) - paketin başlangıcı,

31 34 ("14") - arama düğmesi kodu,

2B ("-") - ilk ayırıcı,

41 31 42 32 43 33 ("A1B2C3") - cihaz kimliği (uzaktan kimlik),

2B ("-") - ikinci ayırıcı,

31 32 33 34 35 36 37 ("1234567") - kullanıcı verileri (SR5-MPRT modeminde düğmeyi kaydederken belirtilir)

03 (ETX) - paket sembolünün sonu.

Python program örneği

Kodun tamamı yazının sonunda verilmiştir.

Bu arada, bu kod metninin tamamını yapay zeka asistanınıza kopyalayın, o da bunu geliştirme ortamınıza mükemmel şekilde uyarlayacaktır.

Gerekli kütüphanelerin yüklenmesi

Kodu çalıştırmadan önce, COM portuyla çalışmak için gerekli kütüphaneleri yüklemeniz gerekir. Yüklemek için aşağıdaki komutu kullanın:

pip install serial-asyncio

Bu kütüphaneler, seri port ile eşzamansız modda çalışma olanağı sağlar.

Değişkenlerin ve port ayarlarının saklanması

Programın başında, seri port ile çalışmak için sabit değerler belirlenir:

```
import asyncio
import serial_asyncio

# COM port settings
SERIAL_PORT = "COM5"  # Specify your port, e.g. "/dev/ttyUSB0" for Linux
BAUDRATE = 9600  # Baud rate
PACKET_LENGTH = 19  # Expected packet length
```

Bu parametreler, bağlantı noktasını, veri hızını ve beklenen paket uzunluğunu bayt cinsinden tanımlar.

SerialReader sınıfı

Gelen verileri işlemek için, aşağıdaki yöntemleri uygulayan SerialReader sınıfı kullanılır:

data_received(self, data): Gelen verileri alır, arabellekte biriktirir ve ayrıştırma için iletir.

process_packet(self, data): Alınan paketi ayrıştırır.

```
class SerialReader(asyncio.Protocol):
    def __init__(self):
        self.buffer = bytearray()
```

```
def data_received(self, data):
    """Process incoming data"""
    self.buffer.extend(data)
    while len(self.buffer) >= PACKET_LENGTH:
        packet = self.buffer[:PACKET_LENGTH]
        self.buffer = self.buffer[PACKET_LENGTH:]
        self.process_packet(packet)  # Pass data to the method

def process_packet(self, data):
    """Parse data packet"""
    if len(data) != PACKET_LENGTH:
        print("Invalid packet length")
        return

    # Extract individual parts of the packet
    start_symbol = chr(data[0])  # First byte - start symbol
    button_number = ''.join(chr(byte) for byte in data[1:3]).strip()  # Bytes 2-3 - button number
    separator1 = chr(data[3])  # 4th byte - separator
    remote_id = ''.join(chr(byte) for byte in data[4:10]).strip()  # Bytes 5-10 - remote ID
    separator2 = chr(data[10])  # 11th byte - second separator
    user_info = ''.join(chr(byte) for byte in data[11:18]).strip()  # Bytes 12-18 - user data
    end_symbol = chr(data[18])  # 19th byte - end symbol

    parsed_data = (
        f'Start symbol: {start_symbol}, Button number: {button_number}, '
        f'Remote ID: {remote_id}, User data: {user_info}, '
        f'Separators: {separator1}, {separator2}, End symbol: {end_symbol}'
    )
    print(parsed_data)
```

Temel ayarlar ve veri işleme sınıfı analiz edildikten sonra, artık portu açma ve veri alma işlemine geçebilirsiniz.

COM portunu açma ve veri alma

Bu örnek, serial_asyncio kütüphanesini kullanarak COM portuyla çalışmanın **eşzamansız yöntemini kullanmaktadır**. Bu yaklaşım size şunları sağlar:

Ana akışı engellemeden verileri gerçek zamanlı olarak işleyin.

Birden fazla bağlantıyla çalışın veya diğer görevleri paralel olarak gerçekleştirin.

Veri beklerken programın donmasını önleyin.

Ancak, program görevi daha basit veya daha tutarlı bir yaklaşım gerektiriyorsa, pyserial kütüphanesiyle **olağan (senkron) yöntemi** kullanabilirsiniz. Örneğin, arka plan işlemleri olmadan az miktarda veriyi işlemeniz gerekiyorsa.

Programı başlattığınızda, belirtilen parametrelerle bir seri port açılır:

```
async def main():
    """Start asynchronous reading from the COM port"""
    print(f'Opening port {SERIAL_PORT} at baud rate {BAUDRATE}')
    loop = asyncio.get_running_loop()
    transport, protocol = await serial_asyncio.create_serial_connection(
        loop, SerialReader, SERIAL_PORT, BAUDRATE
    )
    try:
        await asyncio.Event().wait()  # Infinite wait
    except asyncio.CancelledError:
        pass
    finally:
        transport.close()
        print("Port closed")
```

Paket ayrıştırma

Veriler alındıktan sonra paket ayrıştırılır:

```
  def process_packet(self, data):
    """Parse data packet"""
    if len(data) != PACKET_LENGTH:
      print("Invalid packet length")
      return

    # Extract individual parts of the packet
    start_symbol = chr(data[0])  # First byte - start symbol
    button_number = ''.join(chr(byte) for byte in data[1:3]).strip()  # Bytes 2-3 - button number
    separator1 = chr(data[3])  # 4th byte - separator
    remote_id = ''.join(chr(byte) for byte in data[4:10]).strip()  # Bytes 5-10 - remote ID
    separator2 = chr(data[10])  # 11th byte - second separator
    user_info = ''.join(chr(byte) for byte in data[11:18]).strip()  # Bytes 12-18 - user data
    end_symbol = chr(data[18])  # 19th byte - end symbol

    parsed_data = (
      f'Start symbol: {start_symbol}, Button number: {button_number}, '
      f'Remote ID: {remote_id}, User data: {user_info}, '
      f'Separators: {separator1}, {separator2}, End symbol: {end_symbol}'
    )
    print(parsed_data)
```

Tam örnek kod

```
# EXAMPLE OF ASYNCHRONOUS CONNECTION TO A SERIAL PORT

import asyncio
import serial_asyncio
```

```
# COM port settings
SERIAL_PORT = "COM5"  # Specify your port, e.g. "/dev/ttyUSB0" for Linux
BAUDRATE = 9600  # Baud rate
PACKET_LENGTH = 19  # Expected packet length



class SerialReader(asyncio.Protocol):
    def __init__(self):
        self.buffer = bytearray()

    def data_received(self, data):
        """Process incoming data"""
        self.buffer.extend(data)
        while len(self.buffer) >= PACKET_LENGTH:
            packet = self.buffer[:PACKET_LENGTH]
            self.buffer = self.buffer[PACKET_LENGTH:]
            self.process_packet(packet)  # Pass data to the method

    def process_packet(self, data):
        """Parse data packet"""
        if len(data) != PACKET_LENGTH:
            print("Invalid packet length")
            return

        # Extract individual parts of the packet
        start_symbol = chr(data[0])  # First byte - start symbol
        button_number = ''.join(chr(byte) for byte in data[1:3]).strip()  # Bytes 2-3 - button number
        separator1 = chr(data[3])  # 4th byte - separator
        remote_id = ''.join(chr(byte) for byte in data[4:10]).strip()  # Bytes 5-10 - remote ID
        separator2 = chr(data[10])  # 11th byte - second separator
```

```
        user_info = ''.join(chr(byte) for byte in data[11:18]).strip()  # Bytes 12-18 - user data
        end_symbol = chr(data[18])  # 19th byte - end symbol

        parsed_data = (
            f'Start symbol: {start_symbol}, Button number: {button_number}, '
            f'Remote ID: {remote_id}, User data: {user_info}, '
            f'Separators: {separator1}, {separator2}, End symbol: {end_symbol}'
        )
        print(parsed_data)


async def main():
    """Start asynchronous reading from the COM port"""
    print(f'Opening port {SERIAL_PORT} at baud rate {BAUDRATE}')
    loop = asyncio.get_running_loop()
    transport, protocol = await serial_asyncio.create_serial_connection(
        loop, SerialReader, SERIAL_PORT, BAUDRATE
    )
    try:
        await asyncio.Event().wait()  # Infinite wait
    except asyncio.CancelledError:
        pass
    finally:
        transport.close()
        print("Port closed")


if __name__ == "__main__":
    asyncio.run(main())
```