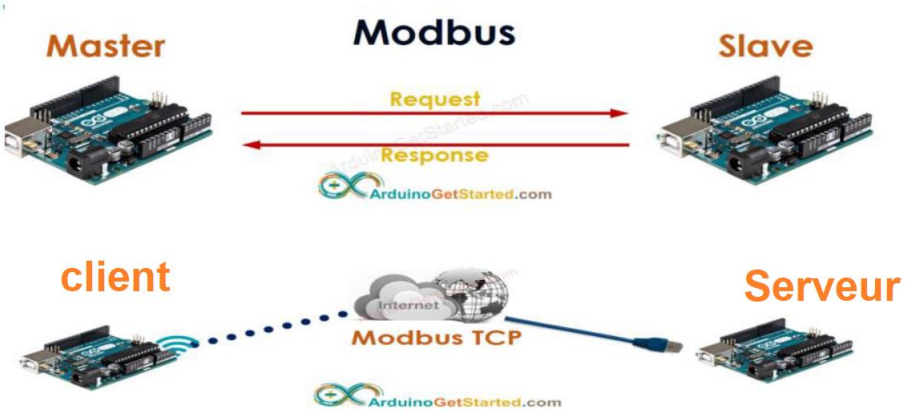
Réseaux Locaux Industriels

TP Examen : Projets sur MODBUS TCP

Background

Les fonctions implémentées par Modbus TCP and Modbus RTU sont les suivantes:

- 01: Read Coils
- 02: Read Discrete Inputs
- 03: Read Holding Registers
- 04: Read Input Registers
- 05: Write Single Coil
- 06: Write Single Register
- 15: Write Multiple Coils
- 16: Write Multiple Registers



ModBus en mode TCP :

Il fonctionne sur le mode client-serveur. En faisant l'analogie avec le mode Master-Slave pour Modbus RTU, slave est le serveur et le Master est le client. Seuls les clients sont actifs, le serveur est complètement passif. C'est le client qui demande des fonctions de lecture et écriture sur le serveur Modbus.
Chaque client doit se connecter au serveur en protocole TCP (adresse IP du serveur, port 502). Le serveur est identifié par : son adresse IP, le numéro du port sur lequel il attend les demandes de connexion (port 502 par défaut).

Deux librairies intéressantes pour le développement d'applications Modbus-TCP avec Python :

- **PyModbus** : https://pymodbus.readthedocs.io/en/dev/readme.html
- **uModbus** : https://umodbus.readthedocs.io/en/latest/

PyModbus

Exemple de code client pour d'écriture et lecture d'un bit dans le serveur. Pour l'écriture le bit est à true (1) et la lecture est dans la variable result.

```
from pymodbus.client.sync import ModbusTcpClient

client = ModbusTcpClient('127.0.0.1')
client.write_coil(1, True)
result = client.read_coils(1,1)
print(result.bits[0])
client.close()
```

read_coils(*address, count=1, **kwargs*)

Parameters:
- **address** – The starting address to read from
- **count** – The number of coils to read
- **unit** – The slave unit this request is targeting

Returns: A deferred response handle

write_coil(*address, value, **kwargs*)

Parameters:
- **address** – The starting address to write to
- **value** – The value to write to the specified address
- **unit** – The slave unit this request is targeting

Returns: A deferred response handle

https://pymodbus.readthedocs.io/en/dev/source/library/pymodbus.client.html?highlight=lient.read_coils#pymodbus.client.common.ModbusClientMixin.read_coils

Plus de détails sur les fonctions **modbus côté client** sont dans :
https://pymodbus.readthedocs.io/en/dev/source/example/synchronous_client.html

C'est un exemple (partie de code) qui permet d'écrire plusieurs registres (8) ayant la valeur par défaut de 20.

```
arguments = {
    'read_address':    1,
    'read_count':      8,
    'write_address':   1,
    'write_registers': [20]*8,
}
log.debug("Read write registeres simulataneously")
rq = client.readwrite_registers(unit=UNIT, **argumen
assert(not rq.isError())     # test that we are not
assert(rq.registers == [20]*8)      # test the expec
```

Exemple de Code Côté serveur :

https://pymodbus.readthedocs.io/en/dev/source/example/synchronous_server.html

```
from pymodbus.device import ModbusDeviceIdentification
from pymodbus.datastore import ModbusSequentialDataBlock, ModbusSparseDataBlock
from pymodbus.datastore import ModbusSlaveContext, ModbusServerContext

from pymodbus.transaction import ModbusRtuFramer, ModbusBinaryFramer

 def run_server():
```

```
store = ModbusSlaveContext(
    di=ModbusSequentialDataBlock(0, [17]*100),
    co=ModbusSequentialDataBlock(0, [17]*100),
    hr=ModbusSequentialDataBlock(0, [17]*100),
    ir=ModbusSequentialDataBlock(0, [17]*100))

context = ModbusServerContext(slaves=store, single=True)

# ------------------------------------------------------------------------- #
# initialize the server information
# ------------------------------------------------------------------------- #
# If you don't set this or any fields, they are defaulted to empty strings.
# ------------------------------------------------------------------------- #
identity = ModbusDeviceIdentification()
identity.VendorName = 'Pymodbus'
identity.ProductCode = 'PM'
identity.VendorUrl = 'http://github.com/riptideio/pymodbus/'
identity.ProductName = 'Pymodbus Server'
identity.ModelName = 'Pymodbus Server'
identity.MajorMinorRevision = version.short()

# ------------------------------------------------------------------------- #
# run the server you want
# ------------------------------------------------------------------------- #
# Tcp:
StartTcpServer(context, identity=identity, address=("", 5020))
```

```
if __name__ == "__main__":
    run_server()
```

La variable **ModbusSlaveContext** est formé par les registres (ici appelés DataBlock) à partir desquels on va écrire ou lire.

`ModbusSequentialDataBlock(0, [17]*100)` : le premier paramètre est l'adresse du début qui est 0. Le deuxième paramètre est la longueur du registre ou block qui est de 100 avec des valeurs initialisés (valeurs par défauts) à 17.

---

uModbus

uModbus ou (µModbus) est une implémentation Python pure du protocole Modbus tel que décrit dans la spécification du protocole d'application MODBUS V1.1b3. uModbus implémente à la fois un client Modbus (à la fois TCP et RTU) et un serveur Modbus (à la fois TCP et RTU).

https://umodbus.readthedocs.io/en/latest/client/tcp.html#example

```
#!/usr/bin/env python
# scripts/examples/simple_tcp_client.py
import socket

from umodbus import conf
from umodbus.client import tcp

# Enable values to be signed (default is False).
conf.SIGNED_VALUES = True

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 502))

# Returns a message or Application Data Unit (ADU) specific for doing
# Modbus TCP/IP.
message = tcp.write_multiple_coils(slave_id=1, starting_address=1, values=[1, 0, 1, 1])

# Response depends on Modbus function code. This particular returns the
# amount of coils written, in this case it is.
response = tcp.send_message(message, sock)

sock.close()
```

https://umodbus.readthedocs.io/en/latest/modbus_server.html

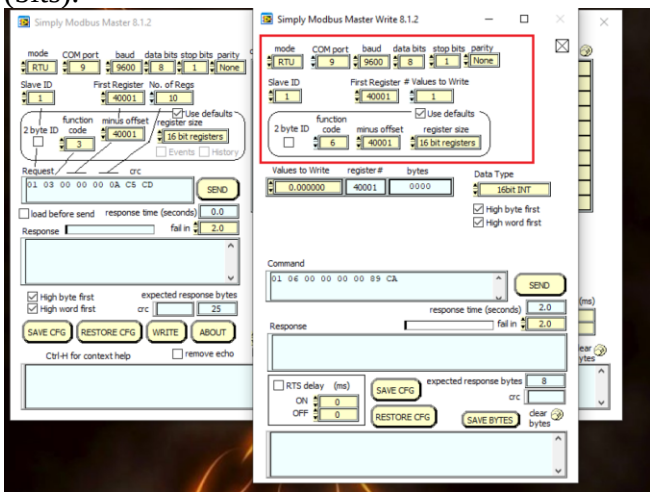
```python
#!/usr/bin/env python
# scripts/examples/simple_tcp_server.py
import logging
from socketserver import TCPServer
from collections import defaultdict

from umodbus import conf
from umodbus.server.tcp import RequestHandler, get_server
from umodbus.utils import log_to_stream

# Add stream handler to logger 'uModbus'.
log_to_stream(level=logging.DEBUG)

# A very simple data store which maps addresss against their values.
data_store = defaultdict(int)

# Enable values to be signed (default is False).
conf.SIGNED_VALUES = True

TCPServer.allow_reuse_address = True
app = get_server(TCPServer, ('localhost', 502), RequestHandler)


@app.route(slave_ids=[1], function_codes=[3, 4], addresses=list(range(0, 10)))
def read_data_store(slave_id, function_code, address):
    """ Return value of address. """
    return data_store[address]


@app.route(slave_ids=[1], function_codes=[6, 16], addresses=list(range(0, 10)))
def write_data_store(slave_id, function_code, address, value):
    """ Set value for address. """
    data_store[address] = value

if __name__ == '__main__':
    try:
        app.serve_forever()
    finally:
        app.shutdown()
        app.server_close()
```

Liste des Projets sur Modbus TCP

Remarque : Il y a 3 projets qui sont multipliés par deux selon qu'on utilise la bibliothèque PyModbus (bibliotéque1) ou uModbus (bibliotéque2). Par exemple le projet 1.1 est celui projet 1 avec l'utilisation de la bibliothèque 1.

	PyModbus	uModbus
Projet 1	**Projet 1.1**	**Projet 1.2**
Projet 2	**Projet 2.1**	**Projet 2.2**
Projet 3	**Projet 3.1**	**Projet 3.2**

Projet 1 : Réaliser une application similaire à 'Simply Modbus Master' qui permet avec une interface graphique de créer des requêtes Modbus en Mode TCP. Pour cela il faut avoir deux applications. Une côté client qui va faire les requêtes et l'autre côté serveur qui va afficher les registres ou les blocks. Il faut faire toutes les fonctions sur les registres et les coils (bits).



Projet 2 : Réaliser une application industrielle qui consiste à lancer la climatisation dans une salle de machine si la température d'un des 3 capteurs de températures soit supérieur à 30 degrés Celsius. On suppose deux modules (programmes) séparés. Un module qui permet de lire les valeurs des capteurs et les transmet vers le deuxième module qui va faire l'action de lancement ou arrêt du climatiseur. Entre les deux modules on utilise la communication Modbus en Mode TCP. Le premier module est considéré comme étant le serveur et le deuxième est considéré comme client. Pour ce projet, on va simuler avec interface graphique la variation des capteurs et le climatiseur.

Projet 3 : Réaliser une application qui permet de convertir une trame de requête de lecture ou écriture Modbus RTU en un requête Modbus TCP. On suppose 2 modules :

- un module Conversion_Modbus_RTU_TCP qui reçoit ou lit les requêtes Modbus RTU et les convertis en un appel de fonction Modbus TCP. Ce module est un client

ModbusTCP. Les requêtes à traiter sont dans un fichier texte où chaque ligne est une requête codée en hexadécimale.

- Un autre module serveur_Modbus_TCP qui réalise les actions du client et affiche le résultat.

module Conversion_Modbus_RTU_TCP

11 03 006B 0003 7687

> ***11***: *The Slave Address (11 hex = address17)*
> ***03***: *The Function Code 3 (read Analog Output Holding Registers)*
> ***006B***: *The Data Address of the first register requested.*
> *(006B hex = 107 , + 40001 offset = input #40108)*
> ***0003***: *The total number of registers requested. (read 3 registers 40108 to 40110)*

module serveur_Modbus_TCP

(affichage des modifications des registres ou réponse sur les valeurs de registres …)