LESSON MC5 – Change Car Status

University of West Attica

Department of Electrical and Electronics Engineering

Ioannis Christidis

Christoforos Kachris

Support by Ethereum Foundation ESP

What will we accomplish!

In this lesson we will create two scripts:

1) A script that periodically checks the status of the car
2) A script that makes a transaction to the blockchain in order to change the car's status

The process will work as follows:

We will first get the renting time from the blockchain. With that we will define the interval between checks on the car's status. For this example, we will go with half the renting time so if the time is 1 hour we will do a check every half hour. With that, we will be sure that we do not miss the time needed to change the car status.

When a check happens, and the car is OCCUPIED, we will calculate the time that the microcontroller should wait before making the transaction to change the status of the car to AVAILABLE. For example, if renting time is 1 hour and we check when the car is already OCCUPIED for 20 minutes, we will wait for 40 minutes to complete the hour and then make a transaction to change status.

Creating `change_car_status.py`

We will start with the easier script which is going to be the `change_car_status.py`.

We will define an `async function` called `change_car_status`.

Since we will be making a transaction, we will need the `address` and the `pk` of the microcontroller.

Then we will wait for the time needed.

```python
async def change_car_status(w3, icr, car_id):
    MC_ADDRESS=os.getenv("MC_ADDRESS")
    MC_PRIVATE_KEY=os.getenv("MC_PRIVATE_KEY")


    remaining_time = get_remaining_time(w3, icr, car_id) # Get the waiting time before making the transaction
    if remaining_time > 0:
        print(f"Waiting {remaining_time + 120} seconds")
        await asyncio.sleep(remaining_time + 120) # Wait for waiting time plus 2 minutes
        # 2 minutes ensure that the block with the correct time has been created
    else:
        print("Renting time is over.")
    print("Starting transaction to change car's status")
    pass
```

We will build `get_remaining_time` later.

Creating change_car_status.py(2)

Next, we make the transaction. We will call the function changeCarStatusMc with the Car ID as input that only the microcontroller of the car is able to call.

```python
async def change_car_status(w3, icr, car_id):
    # … Rest of the logic
    # …

    # Build the transaction
    transaction = icr.functions.changeCarStatusMc(car_id).build_transaction({
        "from": MC_ADDRESS,
        "nonce": w3.eth.get_transaction_count(MC_ADDRESS),
        "gas": 200000,
        "gasPrice": w3.to_wei("20", "gwei"),
    })


    # Sign the transaction
    signed_tx = w3.eth.account.sign_transaction(transaction, MC_PRIVATE_KEY)


    # Send the transaction
    tx_hash = w3.eth.send_raw_transaction(signed_tx.raw_transaction)


    # Wait for the transaction receipt
    receipt = w3.eth.wait_for_transaction_receipt(tx_hash)


    print(f"Transaction successful with hash: {w3.to_hex(tx_hash)}")
    print("Status Changed")
```

Creating get_remaining_time function

The `get_remaining_time` function will work as follows:

1) Get the car and the time it was last rented.
2) Calculate the time that car's status should change.
3) Calculate the time the microcontroller should wait before making the transaction.

```python
def get_remaining_time(w3, icr, car_id):

    car = icr.functions.getCar(car_id).call() # Get the car
    time_of_last_rent = car[3] # From the car get the time where the car was rented

    rent_time = icr.functions.getRentTime().call() # Get the renting time
    time_to_change_status = time_of_last_rent + rent_time # Calculate the time the staus of the car should change

    current_block_timestamp = w3.eth.get_block('latest')['timestamp'] # Get the timestamp of the latest block
    remaining_time = time_to_change_status - current_block_timestamp # Calculate the time the microcontroller should wait

    return remaining_time # Return the time in seconds
```

Creating check_car_status script

Now let's go over creating the check_car_status script.

First of all, we will define a function that retrieves the car data from the SC and checks if the car's status is OCCUPIED. If it is, it will trigger the change_car_status script.

```python
import asyncio
from change_car_status import change_car_status


lock = asyncio.Lock()


async def check_car_status(w3, icr, car_id) :
    async with lock:
        print("Checking car status")
        car = icr.functions.getCar(car_id).call() # Get the car
        status = car[2] # Get the status of the car
        if status == 2: # Check if status is OCCUPIED -> From the Enum Status {0:UNAVAILABLE, 1:AVAILABLE, 2:OCCUPIED}
            print("Car status is Occupied. Enabling Change Car Status")
            await change_car_status(w3, icr, car_id) # Call the change car status
            return True
        else:
            print("Car is not Occupied")
            return False
```

Creating check_car_status script (2)

Next, we will create a function that, every half of renting time, will call the check_car_status function.

```python
async def periodic_check_car_status(rent_time, w3, icr, car_id):
    interval = rent_time/2 # calculate the time per check
    while True:
        if await check_car_status(w3, icr, car_id): # check if status needs update
            continue
        else:
            await asyncio.sleep(interval) # sleep
```

Now let's combine all the scripts in the `main` `function`.

Finishing the `main.py`

Finally let's build the full `main` function. We will use `asyncio.gather` to run our two functions at the same time and give them the necessary parameters.

```
async def main():
    # Load environment variables from .env
    # Check if the .env file exists and has PUBLIC_ADDRESS and PRIVATE_KEY
        # Create an address and private key
        # If account already exists, load it
    # … Rest of the code here
    # Synchronize with ICR Contract
    await synchronize_with_icr()
    load_dotenv(override=True) # Reload after all .env varisables are saved
    w3 = https_connection_to_bc() # Make a connection with the blockchain
    ICR_ADDRESS = os.getenv("ICR_ADDRESS") # Load the ICR Address
    icr_abi = load_json.load_contract_abi("icr_abi.json") # Load the ABI of ICR
    icr = w3.eth.contract(address=w3.to_checksum_address(ICR_ADDRESS), abi=icr_abi) # Make an ICR contract instance
    rent_time = icr.functions.getRentTime().call() # Receive the renting time to calculate the interval for the periodic car status
checks
    CAR_ID = int(os.getenv("CAR_ID")) # Load the car id
    # At the same time
        # Peridically check if car is occupied (and change status to available if needed)
        # Wait for a signed message from a user to unlock the car
    await asyncio.gather(
        periodic_check_car_status(rent_time, w3, icr, CAR_ID),
        handle_signing_for_unlock(w3, icr, CAR_ID)
    )
```

Outro

In the next lesson we will try a full example run of our script.