BlockChain

UNIVERSIDAD PONTIFICIA COMILLAS ICAI

Fundamentos de los Sistemas Operativos

Javier Rojo Llorens (202109117)

José Andrés Ridruejo Tuñón (202111246)

ÍNDICE:

Introducción:

La práctica final de la asignatura de Sistemas Operativos consiste en simular el sistema de validación descentralizado Blockchain, utilizado en sistemas como Bitcoin.

Definición: Blockchain es un libro mayor compartido e inalterable que facilita el proceso de registro de transacciones y de seguimiento de activos en una red de negocios. Un activo puede ser tangible o intangible. Prácticamente cualquier cosa de valor, puede rastrearse y comercializarse en una red de blockchain, reduciendo así el riesgo y los costes para todos los involucrados.

Para el desarrollo de la práctica utilizaremos el programa “Blockchain_app.py’ que llama a la librería ‘Blockchain.py’.

BlockChain.py

La librería ‘Blockchain.py’ contiene las clases ‘Bloque’ y ‘Blockchain’ y sus diferentes funciones asociadas. Construyen el esqueleto principal del proceso.

Bloque:

```
class Bloque:
    def __init__(self,indice,transacciones,timestamp,hash_previo,prueba=0):
        self.indice = indice
        self.transacciones = transacciones
        self.timestamp = timestamp
        self.hash_previo = hash_previo
        self.hash_bloque = None
        self.prueba = prueba
        if not self.hash_bloque:
            self.hash_bloque = self.calcular_hash()
```

Al inicializarla se deben incluir varios parámetros que serán guardados en la clase, al igual que el hash del bloque, el cual se calcula con la función de su mismo nombre.

--------Descripción de la función *calcular_hash(self):*

```
def calcular_hash(self): # calcula el hash de un bloque
    block_string =json.dumps(self.__dict__, sort_keys=True)
    return hashlib.sha256(block_string.encode()).hexdigest()
```

Calcula el hash asociado al bloque en el que estamos trabajando.

--------Descripción de la función *toDict(self):*

```
def toDict(self):
    bloque= {
    'hash_bloque': self.hash_bloque,
    'hash_previo': self.hash_previo,
    'indice': self.indice,
    'timestamp': self.timestamp,
    'prueba': self.prueba,
    'transacciones': self.transacciones }
    return bloque
```

Devuelve el bloque convirtiendo sus variables en valores de un diccionario y lo devuelve como tal.

Blockchain:

```
import hashlib
import json
import time
```

```
class Blockchain(object):
    def __init__(self):
        self.dificultad = 4
        self.transacciones=[]
        self.bloques = [self.primer_bloque()]
```

Se inicia con tres atributos; la dificultad (el numero de ceros que debe contener el inicio del hash para ser valido), las transacciones que todavía no han sido validadas ni integradas en un bloque y el atributo bloques. El atributo bloques contiene la cadena de bloques anclados hasta el momento y se inicia añadiendo un primer bloque con la función *primer_bloque().*

--------Descripción de la función *primer_bloque(self):*

```
def primer_bloque(self):
    primer_bloque = Bloque(1,transacciones =[], timestamp = 0, hash_previo= 1)
    return primer_bloque
```

Se crea un primer bloque que siempre tendrá las mismas características para proporcionar un inicio solido a la cadena.

--------Descripción de la función *nuevo_bloque(self,hash_previo):*

```
def nuevo_bloque(self, hash_previo):
    bloque = Bloque(len(self.bloques)+1,self.transacciones,time.time(),hash_previo)
    return bloque
```

La función crea un bloque basándose en el hash previo, las transacciones no validadas hasta el momento, la longitud de la cadena más uno como el índice y el time.time() como momento en el que se crea.

--------Descripción de la función *nueva_transaccion(self,origen,destino,cantidad):*

```
def nueva_transaccion(self, origen, destino, cantidad):
    transaccion= {"origen": origen,"destino": destino, "cantidad": cantidad,"timestamp": time.time()}
    self.transacciones.append(transaccion)
    return len(self.bloques)+1
```

La función adjunta a la lista de transacciones no validadas una nueva, en modo de diccionario, con las variables “origen”,”destino” y “cantidad”.

--------Descripción de la función *prueba_trabajo(self,bloque):*

```
def prueba_trabajo(self, bloque):
    new_hash = bloque.hash_bloque
    while str(new_hash)[:self.dificultad] != '0'*self.dificultad:
        bloque.prueba += 1
        new_hash =bloque.calcular_hash()
    return str(new_hash)
```

La función encuentra un hash que empiece por el número de ceros requeridos por la dificultad. El proceso se basa en el método de prueba y error y es por lo que se va sumando de una unidad en una unidad al atributo prueba

--------Descripción de la función *prueba_valida(self,bloque,hash_bloque):*

```
def prueba_valida(self, bloque, hash_bloque):
    if hash_bloque[:self.dificultad] == '0'*self.dificultad:
        if bloque.calcular_hash() == hash_bloque:
            return True
    return False
```

La función comprueba la veracidad del hash del bloque calculando el número de ceros por los que empieza la cadena y viendo si calculándolo da lo que debería.

--------Descripción de la función *integra_bloque(self,bloque_nuevo,hash_prueba):*

```python
def integra_bloque(self, bloque_nuevo, hash_prueba):
    if self.prueba_valida(bloque_nuevo,hash_prueba):
        if bloque_nuevo.hash_previo == self.bloques[-1].hash_bloque:
            self.bloques.append(bloque_nuevo)
            bloque_nuevo.hash_bloque = hash_prueba
            bloque_nuevo.transacciones = self.transacciones
            self.transacciones = []
            return True
    return False
```

La función hace todas las comprobaciones para asegurar que el bloque es válido en la cadena y lo integra, asociándole las transacciones y su hash de prueba.

--------Descripción de la función *toDict(self):*

```python
def toDict(self):
    chain = []
    for bloque in self.bloques:
        chain.append(bloque.toDict())
    return {'dificultad': self.dificultad, 'transacciones': self.transacciones, 'chain': chain}
```

La función convierte toda la información del blockchain en un diccionario, incluyendo, a parte de los bloques, la dificultad y las transacciones no añadidas a la cadena.

Blockchain_app.py

Este archivo maneja la creación de la cadena interactuando con la “app” que creamos

```
import BlockChain       peperidruejo, yesterday • main
from threading import Semaphore ,Thread
import time
import socket
from flask import Flask, jsonify, request
from argparse import ArgumentParser
from datetime import datetime
import json
import platform

# semaforo para no pisarse con el escritor de la cader
# y el escritor de la cadena en archivo json
backup = Semaphore(1)

# Instancia del nodo
app =Flask(__name__)

# Instanciacion de la aplicacion
blockchain =BlockChain.Blockchain()
nodos_red = set()

# Para saber mi ip
mi_ip =socket.gethostbyname(socket.gethostname())
```

Se importan las librerías así como BlockChain.py, se crea un semáforo que evita la solapación entre la escritura de la cadena en archivo json y añadir un bloque a la cadena.

--------Descripción de la función *nueva_transaccion():*

```
def nueva_transaccion():
    values =request.get_json()
    # Comprobamos que todos los datos de la transaccion estan
    required =['origen', 'destino', 'cantidad']
    if not all(k in values for k in required):
        return 'Faltan valores', 400
    # Creamos una nueva transaccion aqui
    blockchain.nueva_transaccion(values['origen'], values['destino'], values['cantidad'])
    index = len(blockchain.bloques) + 1 # la transaccion se guardará en el bloque siguiente (+1)
    response ={'mensaje': f'La transaccion se incluira en el bloque con indice {index}'}
    return jsonify(response), 201
```

En el caso de que estén todos los required arguments, se guarda la nueva transacción y se devuelve el índice del bloque en el que se añadirá a la cadena.

--------Descripción de la función *blockchain_completa():*

```
def blockchain_completa():
    response ={
    # Solamente permitimos la cadena de aquellos bloques finales que tienen hash
    'chain': [b.toDict() for b in blockchain.bloques if b.hash_bloque is not None],
    }
    response['longitud']= len(response['chain'])
    return jsonify(response), 200
```

Devuelve la cadena completa y la longitud de la misma como un diccionario.

--------Descripción de la función *detallesnodo():*

```
def detallesnodo():
    detalles = {'maquina':platform.machine(),'nombre_sistema':platform.system(),'version':platform.vesion()}
    return detalles
```

La función devuelve algunos detalles del nodo que está ejecutando

--------Descripción de la función minar*():*

```python
def minar():
    global backup,mi_ip

    # No hay transacciones
    if len(blockchain.transacciones) == 0:
        response ={
        'mensaje': "No es posible crear un nuevo bloque. No hay transacciones"
        }
    else:
        # se añade la transaccion del minero
        blockchain.nueva_transaccion('0', mi_ip, 1)
        # se obtiene el hash previo
        hash_previo = blockchain.bloques[-1].hash_bloque
        # nuevo bloque
        new = blockchain.nuevo_bloque(hash_previo)
        # nuevo hash
        new_hash = blockchain.prueba_trabajo(new)
        # para insertar el bloque tenemos que no solaparnos con el json que escribe la cadena cada 60s
        backup.acquire()
        if blockchain.integra_bloque(new,new_hash):
            response ={
            'mensaje': "Nuevo bloque minado \n"+ str(new.toDict())
            }
        backup.release()
    return jsonify(response), 200
```

Para la función minar, se buscan las transacciones no actualizadas. Si no hay, no se puede minar nada, si hay, se añade la transacción del minero en cuestión y se busca el hash asociado a ese nuevo bloque. Al encontrarlo, se integra todo a la blockchain.

--------Descripción de la función copia_de_seguridad*():*

```python
def copia_de_seguridad(): # copia de seguridad que se eje
    global backup
    while True:
        time.sleep(60)
        backup.acquire()
        data = {}
        data['chain'] = [blockchain.bloques]
        data['longitud'] = len(blockchain.bloques)
        now = datetime.now()
        dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
        data['date'] = dt_string
        with open('data.json', 'w') as file:
            json.dump(data, file, indent=4)
        backup.release()
```

Esta función, manejada por un hilo, crea una copia de seguridad cada 60 s en la cual se guarda toda la cadena en un archivo de tipo JSON.

--------Descripción de la función registrar_nodos_completo()

```python
def registrar_nodos_completo():
    values =request.get_json()
    global blockchain
    global nodos_red
    global puerto
    nodos_nuevos =values.get('direccion_nodos')
    if nodos_nuevos is None:
        return "Error: No se ha proporcionado una lista de nodos", 400
    all_correct =True
    for nodo in nodos_nuevos:
        # almacenar los nodos recibidos en nodos_red y enviará a dichos nod
        nodos_red.add(nodo)
        lista_nodos = list((nodos_red-{nodo}) | {f'http://{mi_ip}:5000'})
        blockchain_2 = blockchain.toDict()
        data = {'chain': blockchain_2, 'nodos': lista_nodos}
        response = requests.post(f'{nodo}/nodos/actualizar', json=data)
```

La función recibe una serie de nodos que añade a los nodos de la red y les actualiza enviándoles los datos de la cadena convertidos en Diccionario.

--------Descripción de la función registrar_nodo_actualiza_blockchain*():*

```python
def registrar_nodo_actualiza_blockchain():
    global blockchain
    read_json =request.get_json()
    nodes_addreses =read_json.get("nodos_direcciones")

    if nodes_addreses is None:
        response={
        'mensaje': 'Error: No se ha proporcionado una lista de nodos'
        }
        return jsonify(response), 400

    for node in nodes_addreses:
        nodos_red.add(node)

    blockchain_2 = read_json.get("chain")
    if blockchain_2 is None:
        response={
        'mensaje': 'Error: No se ha proporcionado una blockchain'
        }
        return jsonify(response), 400

    blockchain = blockchain.fromDict(blockchain_2)


    response={
    'mensaje': 'Se ha actualizado la blockchain',
    'nodos_totales': list(nodos_red)
    }
    return jsonify(response), 200
```

La función pasa la copia del blockchain a los nuevos nodos.