Implementação de Threads e Semáforos em Python:

A biblioteca "threading", vem nativamente no python e nos permite implementar *threads* e *semáforos* de forma simples e intuitiva.

A seguir uma demonstração da biblioteca, utilizando-a para resolver problemas clássicos de concorrência:

---

1. Problema Leitor/Escritor:

```
from random import randint
from threading import Thread, Semaphore # Importando as classes Thread
import time
from Relogio import Relogio # Classe de cronômetro/timer

podeLer = Semaphore(1) # Semáforo inicializado com valor 1, indica se a

podeEscrever = Semaphore(0) # Semáforo inicializado com valor 0, indica

listaDisponivel = Semaphore(1) # Semáforo inicializado com valor 1, ind

class Leitor(Thread): # Classe Leitor, filha da classe Thread
    def __init__(self, lista): # Inicializando a classe filha
        Thread.__init__(self) # Inicializando a classe pai
        self.lista = lista # Passando lista como parâmetro

    def run(self): # Função run é onde deve ser colocado o código princ

        while True:

            # Método acquire decrementa 1 do valor passado para o semáf
            podeLer.acquire() # Menos pode ser lido
            listaDisponivel.acquire() # Lista está indisponível no mome

            soma = 0
            for i in self.lista: soma += i # Soma todos os itens da lis
            print(soma)
            time.sleep(1)

            # Método release adiciona 1 ao valor passado para o semafor
            listaDisponivel.release() # Lista já pode ser utilizada
            podeEscrever.release() # Lista já pode ser escrita

class Escritor(Thread): # Classe Escritor, filha da classe Thread
```

```
    def __init__(self, lista):
        Thread.__init__(self)
        self.lista = lista

    def run(self):
        while True:

            podeEscrever.acquire() # Menos um pode ser escrito
            listaDisponivel.acquire() # Lista indisponível no momento

            self.lista.pop(0) # Retira o primeiro elemento da lista
            self.lista.append(randint(0,10)) # Adiciona número aleatóri
            time.sleep(1)
            print(self.lista)

            listaDisponivel.release() # Lista já pode ser utilizada
            podeLer.release() # Lista já pode ser lida

def main():
    time = Relogio(10) # O programa irá rodar por 10 segundos
    time.start()

    lista = [1,2,3]
    print(lista)

    leitor1 = Leitor(lista)
    leitor1.start()

    leitor2 = Leitor(lista)
    leitor2.start()

    escritor1 = Escritor(lista)
    escritor1.start()

    escritor2 = Escritor(lista)
    escritor2.start()

main()
```

2. Output:

```
[1, 2, 3]
6
[2, 3, 4]
9
[3, 4, 10]
17
[4, 10, 3]
17
[10, 3, 10]
23
```

3. Problema Produtor/Consumidor:

```
from threading import Thread
from threading import Semaphore
from Relogio import Relogio
import time
import random

lista = []
podeProduzir = Semaphore(10) # Indica que 10 podem ser produzidos
podeConsumir = Semaphore(0) # Indica que 0 podem ser consumidos inicial
listaDiponivel = Semaphore(1) # Indica se a lista está sendo utilizada

class Produtor(Thread):
    def __init__(self, lista, num):
        Thread.__init__(self)
        self.lista = lista
        self.num = num

    def run(self):
        while True:
            podeProduzir.acquire() # Produziu 1, subtrai 1 dos que ainc
            listaDiponivel.acquire() # Lista está sendo utilizada, rese

            self.lista.append(random.randint(0, 10))# Adiciona número a
            print(f"Produtor {str(self.num)} adicionou {str(self.lista|
            time.sleep(0.5)

            listaDiponivel.release() # Terminado o processo, lista está
            podeConsumir.release() # Mais 1 pode ser consumido

class Consumidor(Thread):
    def __init__(self, lista, num):
        Thread.__init__(self)
        self.lista = lista
        self.num = num

    def run(self):
        ultimo = None
        while True:
            if len(lista) == 10:break
            podeConsumir.acquire() # Menos 1 pode ser consumido
            listaDiponivel.acquire() # Lista está sendo utilizada, rese

            ultimo = self.lista.pop(0) # Retira o primeiro item da list
            print(f"Consumidor {str(self.num)} retirou {str(ultimo)}: {
            time.sleep(0.5)

            listaDiponivel.release() # Lista já pode ser utilizada
```

```
            podeProduzir.release() # Mais 1 item pode ser produzido


def main():
    timer = Relogio(10) # O programa vai executar por 10 segundos
    timer.start()

    produtor1 = Produtor(lista, 1)
    produtor1.start()

    consumidor1 = Consumidor(lista, 1)
    consumidor1.start()

    produtor2 = Produtor(lista, 2)
    produtor2.start()

    consumidor2 = Consumidor(lista, 2)
    consumidor2.start()

    exit()

main()
```

4. Output:

```
Produtor 1 adicionou 9: [9]
Produtor 1 adicionou 4: [9, 4]
Produtor 1 adicionou 2: [9, 4, 2]
Produtor 1 adicionou 9: [9, 4, 2, 9]
Produtor 1 adicionou 0: [9, 4, 2, 9, 0]
Produtor 1 adicionou 8: [9, 4, 2, 9, 0, 8]
Produtor 1 adicionou 1: [9, 4, 2, 9, 0, 8, 1]
Produtor 1 adicionou 1: [9, 4, 2, 9, 0, 8, 1, 1]
Produtor 1 adicionou 1: [9, 4, 2, 9, 0, 8, 1, 1, 1]
Consumidor 1 retirou 9: [4, 2, 9, 0, 8, 1, 1, 1]
Consumidor 1 retirou 4: [2, 9, 0, 8, 1, 1, 1]
Consumidor 1 retirou 2: [9, 0, 8, 1, 1, 1]
Consumidor 1 retirou 9: [0, 8, 1, 1, 1]
Consumidor 1 retirou 0: [8, 1, 1, 1]
Consumidor 1 retirou 8: [1, 1, 1]
Consumidor 1 retirou 1: [1, 1]
Consumidor 1 retirou 1: [1]
Produtor 2 adicionou 8: [1, 8]
Produtor 2 adicionou 4: [1, 8, 4]
Produtor 2 adicionou 4: [1, 8, 4, 4]
```

---

5. Problema do Jantar dos Filósofos:

```
from threading import Thread, Semaphore
```

```
import random
import time

class Filosofo(Thread):
    comendo = True # Indica se todos estão comendo
    def __init__(self, nome,  index, garfoEsquerdo, garfoDireito):
        Thread.__init__(self)
        self.nome = nome
        self.index = index
        self.garfoEsquerdo = garfoEsquerdo
        self.garfoDireito = garfoDireito

    def run(self):
        while self.comendo:
            time.sleep(random.randint(1,5)) # Tempo inicial que o filós
            print(f'Filósofo {self.nome} quer comer.')
            self.comer() # Filósofo começa a comer.

    def comer(self):
        # Se ambos os semáforos (talheres) estiverem livres, o filósofo
        esquerdo, direito = self.garfoEsquerdo, self.garfoDireito

        while self.comendo:
            esquerdo.acquire() # Indica que o garfo à sua esquerda não

            disponivel = direito.acquire(False) # Verifica disponibilid
            if disponivel:
                break

            # Caso o garfo direito não esteja disponível, solta o garfo
            esquerdo.release()

            # Filósofo troca os talheres
            print(f'Filósofo {self.nome} troca os talheres.')
            esquerdo, direito = direito, esquerdo
        else:
            return

        print(f'Filósofo {self.nome} come.')
        time.sleep(5)
        print(f'Filósofo {self.nome} começa a pensar.')

        # Solta os dois talheres depois de comer
        direito.release()
        esquerdo.release()

def main():
    # Inicializando vetor de semáforos(talheres)
    talheres = [Semaphore() for n in range(5)]

    # A divisão modular -> (i+1)%5 é usada para pegar circularmente os
    nome=["Sócrates", "Platão", "Aristóteles", "Epíteto", "Tales"] # No
```

```
    filosofos = [Filosofo(nome[i], i, talheres[i % 5],
                talheres[(i+1) % 5])
                for i in range(5)
                ]

    # Inicializa as threads de cada filósofo
    Filosofo.comendo = True
    for filosofo in filosofos:
        filosofo.start()
    time.sleep(10)
    Filosofo.comendo = False

    print("Todos comeram.")

main()
```

6. Output:

```
Filósofo Platão quer comer.
Filósofo Platão come.
Filósofo Aristóteles quer comer.
Filósofo Tales quer comer.
Filósofo Tales come.
Filósofo Sócrates quer comer.
Filósofo Epíteto quer comer.
Filósofo Epíteto troca os talheres.
Filósofo Platão começa a pensar.
Filósofo Aristóteles come.
Filósofo Tales começa a pensar.
Filósofo Epíteto troca os talheres.
Filósofo Sócrates come.
Filósofo Platão quer comer.
Todos comeram.
```

> Sistemas Operacionais I
>
> Luís Henrique Scantelbury de Almeida
>
> Luã Maury Maquiné da Silva
>
> Arthur Lucas dos Santos Bezerra
>
> 17/11/2021