Sincronização de Processos com Semáforos

Com a finalidade de garantir a sincronização entre os processos que acessam recursos do sistema de forma simultânea é necessário usar técnicas para verificar e controlar o acesso aos dados, a fim de evitar inconsistência de dados ou que processos não fiquem em espera por muito tempo.

A técnica utilizada para a solução do problema proposto foi Semáforo que resolve o problema de exclusão mútua. O semáforo sofre dois tipos de operação sobre ele que são permitindo o acesso e bloqueando o acesso das Threads.

A função adquir() quando executada por uma thread bloqueia a seção crítica e o count da função é decrementado em 1, assim que a thread executar a função release() a seção crítica será liberada e o count é incrementado em 1, dessa forma uma próxima thread pode entrar na seção crítica é utilizar os recursos.

As operações de incrementar e decrementar devem ser operações atômicas, ou seja, enquanto um processo estiver executando uma dessas duas operações, nenhum outro processo pode executar outra operação sob o mesmo semáforo, devendo esperar que o primeiro processo encerre a sua operação sob o semáforo. Essa obrigação evita condições de disputa entre vários processos.

```python
from threading import Thread #Importação da class Thread
from threading import Semaphore #Importação da class Semaphore
from time import sleep #Importação da funcão sleep

contador = 0
obj = Semaphore() #A classe Semaphore faz o controle de threads que podem acessar um recurso simultaneamente

class Processo(Thread):  #Class Processo
    def __init__(self,p): #Construtor da classe Processo
        Thread.__init__(self) #Construtor da classe Thread
        self.p=p

    def run(self): #Método contendo o que será executado por cada thread

        global contador

        while True:
            obj.acquire() #Este método realiza o procedimento de bloqueio da região crítica (semáforo bloqueado)
            print(f'Processo: {self.p}, entrou na região crítica, contador: {contador}')
            contador += 1 #Mostra a sequêcia de entrada e saída dos processos
            print(f'Processo: {self.p}, saiu na região crítica, contador: {contador}')
            obj.release() #Este método desbloqueia a região crítica (semáforo liberado)
            sleep(1)

for i in range(10): #for responsável por criar 10 threads e inicializa-lás
    proc = Processo(i)
    proc.start()
```