THREADS NA PRÁTICA COM PYTHON

Raimundo Angeliano Gonçalves de Sousa

Trabalho da Cadeira de Sistemas Operacionais

Profº Rafael Gomes Lopes

FORTALEZA - CE 2022

Descrição do Projeto

O objetivo do projeto é aprofundar os estudos com threads aplicando em um exercício para cumprir os seguintes requisitos:

- Gerar um vetor de 10^8 posições preenchido aleatoriamente com algum dígito entre 0 a 9.
- Armazenar quantas vezes aparece cada dígito possível.
- Exibir em tempo real a contagem de quantos dígitos foram encontrados até o momento.
- Permitir escolher quantas threads vão realizar a tarefa, variando entre 1, 2, 5 e 10.
- Abordagem de como organizar as threads para realizar a busca é da equipe.
- Garantir exclusão mútua usando as estratégias de semáforo e trava.
- Repetir cada caso possível executar 30 vezes o experimento, ou seja, para os 10 casos existentes com a combinação número de threads e estratégia de exclusão mútua.
- Descrever no Documento a ser entregue a comparação da média de duração da execução em cada caso possível (número de threads e estratégia de exclusão mútua).

1. Implementação

Para cumprir com os requisitos foram adicionadas os seguintes classes e funções.

 # Classe

- **counter thread:** Essa classe realiza a contagem dos números armazenados em um dicionário de forma assincrona e para evitar problemas com concorrência está sendo usado a técnica de trava e semaforo para evitar que mais de uma thread acesse a região crítica.

 # Funções

- **pop vector without thread:** Função para popular o **vetor referência** (vetor que armazena os números)
- **print vector count:** O objetivo é mostrar a contagem do nosso **vetor contagem** (vetor que conta quantos números de 0 a 9 foram armazenados no vetor referência).
- **count numbers without thread:** Função para realizar a contagem sem precisar de uma thread, também utilizada no primeiro caso.
- **incremento dictionary:** Essa função é responsável por acessar o vetor contagem e somar a posição no nosso dicionário.

2. Bibliotecas Utilizadas

- **Threading:** Para a criação e gerenciamento de threads e controle da região crítica, com as funções **Lock** e **Thread**
- **Time:** Necessário para fazer a contagem do tempo de execução desde o início que a thread inicia até o seu final.
- **Random:** Basicamente estou usando a função **randint** para preencher com um inteiro aleatório o nosso **vetor referência**

3. Resultados Obtidos

Conforme o último requisito, para realizar um comparativo mais fiel entre o uso das threads o processo foi repetido **30 vezes** e calculado a média do tempo de execução em cada caso

- **01 Thread(s):** 1010,656288875 segundos
- **02 Thread(s):** 1671,107365855 segundos
- **05 Thread(s):** 1526.179496765 segundos
- **10 Thread(s):** 2229.949999411 segundos

Considerações Finais

Como podemos observar por mais que sejam implementadas threads sempre vamos ter o problema da concorrência influenciando no tempo de execução do projeto e no seu resultado final, nesse projeto foi utilizado a técnica de semáforo e travas para impedir que mais de uma thread altere o vetor contagem ao mesmo tempo.

Então utilizando as técnicas de trava e semáforo conseguimos garantir que pelo menos o resultado final é mais confiável.

Referências Bibliográficas

- https://www.tutorialspoint.com/python3/python_multithreading.htm
- https://realpython.com/intro-to-python-threading/
- https://docs.python.org/3/library/threading.html#semaphore-objects
- https://docs.python.org/3/library/time.html
- https://docs.python.org/3/library/random.html
- **Projeto no github:** https://github.com/angelianosousa/Estudando-SO/tree/master/Threads