UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS, TECNOLOGIAS E SAÚDE
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Lista de Exercícios 1

Bernardo Pandolfi Costa

Disciplina: Sistemas Operacionais
Professor: Roberto Rodrigues Filho

Araranguá
11 de outubro de 2023

Conteúdo

1 Exercício 1

O programa pede ao usuário um comando a ser executado e depois o executa. Exemplos:

- ls .
- ls . -la
- touch . hello

2 Exercício 2

Para cada processo, o A manteve um valor diferente. Quando A é colocado em cada processo, a variável global é copiada com valor não alterado, neste caso, $A = 1$. Como cada processo é isolado, a alteração feita no processo filho não é carregada ao processo pai, de forma que a variável A terá um valor diferente em cada processo, baseado na alteração de cada um.
No código, foi programado que o processo filho executa uma multiplicação de 10 em A, enquanto o processo pai multiplicará A por -10, para que a mudança seja mais perceptível. Assim, no processo filho o resultado final de A foi de 10 e o do processo pai foi de -10.

3 Exercício 3

Cada *thread* executa ao mesmo tempo, mas seus valores são compartilhados, visto que a variável A é global. Então as alterações que são feitas em cada *thread* executa sua função independentemente das outras, mas os resultados são compartilhados.
Neste código, foi programado que a *thread* principal (dentro do main) multiplica A por 10 e a *thread* secundária multiplicará por -10, de forma similar ao exemplo anterior. Para facilitar a visualização, foi adicionado um atraso na multiplicação de -10 para que seja perceptível que a multiplicação de -10 na *thread* secundária é feita **após** a multiplicação da *thread* principal, apesar de ocorrerem independentemente.

4 Exercício 4

No código, através das bibliotecas *BeautifulSoup* e *requests*, foi possível acessar os conteúdos de uma página na web sem a necessidade de salvá-la na máquina. Assim, o programa se conecta ao *url* pedido, consegue o corpo do HTML e busca a quantidade de aparições da palavra especificada e retorna o valor ao terminal.

5 Exercício 5

De forma similar ao exercício anterior, as bibliotecas *BeautifulSoup* e *requests* foram utilizadas. Neste caso, a *String* que contém os conteúdos da página é divida em

três partes de tamanhos iguais e, na criação das *threads*, cada uma recebe uma parte da *String*. Assim, cada *thread* buscará a quantidade de vezes que a palavra aparece em sua respectiva seção do HTML, somando as aparições concomitantemente à mesma variável.

6 Exercício 6

O código busca o conteúdo do body da página e o coloca em uma grande *String*, chamada de *body_text* no código. Em seguida, a variável *body_text* é divida em três outras de tamanhos iguais, para que cada pedaço seja colocado em uma *thread* específica. Antes das *threads* serem criadas, uma variável que carrega o momento de início da operação *multithread* é instanciada pelo método *time()*. Ao fim do processo, outra variável que indica o momento em que a operação chega ao fim é criada. O resultado de tempo Δt, então, equivale à subtração do tempo final menos o tempo inicial, que é mostrado no terminal.
Ao fim do processo *multithread*, a operação é repetida, mas desta vez, com apenas uma *thread*. Os resultados indicam, então, que o programa em *monothread* é de 3 a 4 vezes mais rápido que o código em *multithread*. Existem alguns possíveis motivos para este resultado. Como instanciar e gerenciais *threads* demanda tempo e poder computacional, quando estamos lidando com tarefas simples, como contar as ocorrências de uma palavra, pode ser que a estratégia não compense.
Outra explicação ao ocorrido se deve a como códigos em Python são interpretados pelo GIL (Global Interpreter Lock) em CPython. Este interpretador executa o código em apenas um processo e faz com que os *threads* do código não sejam verdadeiramente paralelos como teoricamente deveriam.