2019/20

Programação Paralela e Distribuída

Prática 1: Python Threading

https://www.tutorialspoint.com/python/python_multithreading.htm
https://docs.python.org/3.8/library/threading.html

1 – Implemente o exemplo Hello_World.py da aula teórica 2.

2 – Modifique o programa anterior para criar duas threads. As mensagens devem agora indicar o nome da thread que lhes deu origem.

3 – Resolva em Python o exercício proposto na página 70 da aula teórica 2.

a) Comece por calcular sequencialmente a soma de um array numpy com os valores [1..1000].

b) Construa uma solução em que uma thread calcula a soma da primeira metade do array e outra thread calcula a soma da segunda metade do array. Use uma solução em que define uma função para a thread.

c) Construa agora uma solução em que define a thread como subclasse de Thread.

Nota: Para os exercícios que se seguem tem uma solução em java, em https://www.di.ubi.pt/~pprata/spd/SD_16_17_T03b.pdf)

Construa soluções em python:

4 – Suponha duas threads **p1** e **p2** que partilham uma variável comum, **variavelPart**. Veja a estrutura do código das duas threads na página seguinte. Pretende-se construir um exemplo que ilustre a violação de uma secção crítica, sem usar qualquer tipo de mecanismo de sincronização.

- Considere que o processo p1 possui duas variáveis locais, x e y, inicializadas com valores simétricos, e que dentro de um ciclo infinito transfere a quantidade armazenada em **variavelPart** de x para y. O processo 2 vai, em cada iteração, incrementar a variável partilhada.

Pretende-se que a condição x + y = 0 seja verdadeira durante toda a execução do programa. Quando, no processo p1, se detecta que a secção crítica foi violada (porque x + y **!=** 0) o processo deve terminar e acabar o programa. Para isso transforme P2 numa thread daemon.

Suponha que P1 e P2 têm a estrutura:

Processo 1	**Processo 2**

x = M; y = - M;
While (true){
//**secção crítica 1**
x = x – variavelPart;
y = y + variavelPart;
<**parte restante** 1>
if (x+y != 0){
print "Secção crítica violada"
break;
}//fim do if
…
}// fim do While
	...
	While (true){
	//**secção crítica 2**
	variavelPart =
	variavelPart +1;
	<**parte restante** 2>
	}

5 – Altere o programa anterior com um mecanismo de sincronização que garante que a secção crítica não será violada. Explore os mecanismos lock, semáforo, e event.

6 – Considere o **"Readers-Writers Problem"**:

a) Construa uma classe em RW, que possua um campo inteiro, XPTO, e dois métodos: ler e escrever. O método ler deve devolver o valor da variável XPTO; o método escrever deve adicionar o valor 100 à variável XPTO e seguidamente subtrair o mesmo valor à variável XPTO.

b) Pretende-se que um objecto da classe RW seja partilhado por vários processos (Threads) de dois tipos:

- processos Leitores – que lêem o valor da variável XPTO usando o método ler;

- processos Escritores – que alteram a variável XPTO usando o método escrever.

- Construa as classes Leitor e Escritor. Cada uma destas classes deve ter uma Thread de execução própria em que, num ciclo infinito, vão respectivamente lendo e alterando valores do objecto partilhado.

c) Construa uma classe de teste que crie um objecto do tipo RW, 3 objectos do tipo Leitor e 2 objectos do tipo Escritor. Estude o comportamento do seu programa

d) Pretende-se que modifique as classes anteriores tal **que os vários processos Leitores possam executar concorrentemente o método ler, mas que quando um processo Escritor executar o método escrever o faça em exclusão mútua**. Isto é, quando um processo está a escrever, nenhum outro pode em simultâneo ler ou escrever a variável XPTO.