

Roteiro 2 - Comunicação Interprocessos

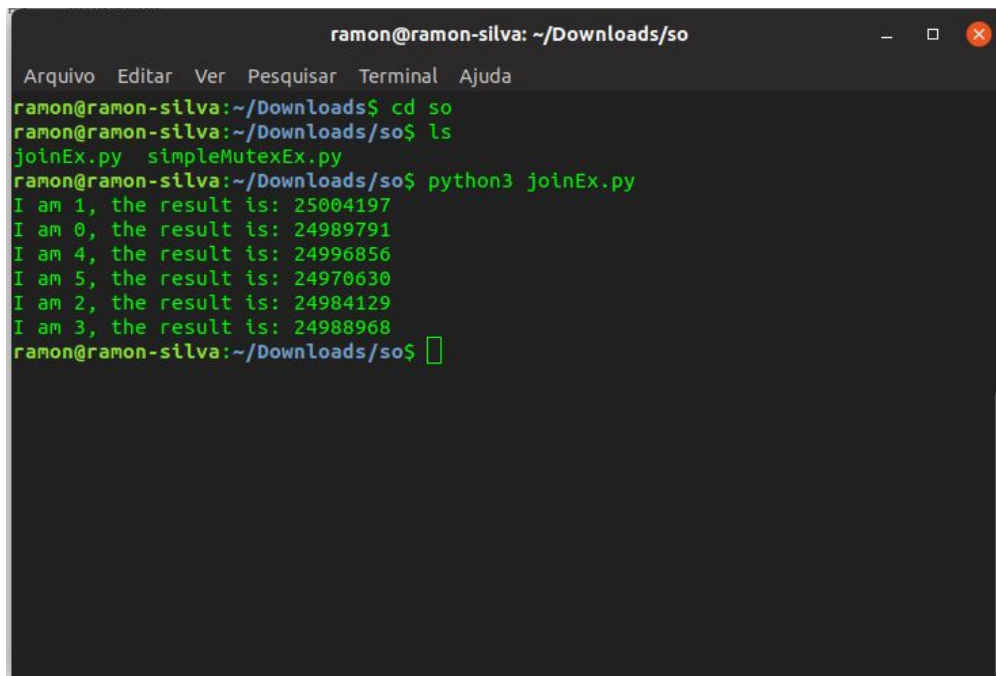
5. Classe Thread: A classe Thread cria a thread, de forma separada da thread de controle

thread.start(): Inicia o Thread, executa separadamente do thread de controle.

thread.join(): Quando o join é chamado ele bloqueia o thread principal até que a execução do processo que chamou o *join()* termine.

Obs. Caso seja feito uma tentativa de entrar num thread em execução, vai ser gerado o erro de `RuntimeError`.

6.



```
ramon@ramon-silva: ~/Downloads/so
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
ramon@ramon-silva:~/Downloads$ cd so
ramon@ramon-silva:~/Downloads/so$ ls
joinEx.py  simpleMutexEx.py
ramon@ramon-silva:~/Downloads/so$ python3 joinEx.py
I am 1, the result is: 25004197
I am 0, the result is: 24989791
I am 4, the result is: 24996856
I am 5, the result is: 24970630
I am 2, the result is: 24984129
I am 3, the result is: 24988968
ramon@ramon-silva:~/Downloads/so$
```

7. Porque quem está responsável pela criação dos threads é o sistema operacional, e quem decide quem vai ser criado e o momento em que vai ser criado é o escalonador do SO.

8. Não acontece nada, não necessita da utilização do *join()* para a obtenção dos resultado.

9. Finaliza a Thread, quando a mesma termina suas operações.

11. Classe Lock: A classe que implementa objetos de bloqueio primitivos. Depois que um thread adquire um bloqueio, as tentativas subseqüentes de adquiri-lo são bloqueadas até que seja liberado.

Lock.acquire(): Bloqueia outros processos até que seja solicitado o desbloqueio. Devido estar em sua região crítica.

Lock.release(): Desbloqueia os threads que estão no estado de bloqueio.

12.

```
ramon@ramon-silva: ~/Downloads/so
Arquivo Editar Ver Pesquisar Terminal Ajuda
ramon@ramon-silva:~$ cd Downloads/so
ramon@ramon-silva:~/Downloads/so$ ls
joinEx.py  simpleMutexEx.py
ramon@ramon-silva:~/Downloads/so$ python3 simpleMutexEx.py
1000 added!
1000 added!
1000 added!
1000 added!
1000 added!
Shared data = 5000
ramon@ramon-silva:~/Downloads/so$
```

1. classe Lock
2. Lock.acquire()
3. Lock.release()
11. Dê uma descrição do que faz cada um desses tipos ou funções.
12. Rode o programa. O que ele faz?
14. Desenvolva um programa em Python que modela o problema do produtor-consumidor usando threads e as diretivas de sincronização de Python. Seu programa deve ter as seguintes características:
 1. O tamanho do buffer deve estar definido em uma constante declarada logo após os imports. Eu quero poder mudar o tamanho do buffer e o seu programa deve continuar funcionando corretamente.
 2. Sempre que o produtor produzir um item X (que deve ser um inteiro), ele deve escrever na tela "Produced X"
 3. Sempre que o consumidor consumir um item X (que deve ser um inteiro), ele deve escrever na tela "Consumed X"
 4. Faça uma função de delay para podermos ver o que acontece no terminal

Com as diretivas *acquire()* e *release()* o thread incrementa +1000, na variável global "Share data" sendo seu resultado final igual a 5000.

13.

```
ramon@ramon-silva: ~/Downloads/so
Arquivo Editar Ver Pesquisar Terminal Ajuda
ramon@ramon-silva:~/Downloads/so$ ls
joinEx.py  simpleMutexEx.py
ramon@ramon-silva:~/Downloads/so$ python3 simpleMutexEx.py
1000 added!
1000 added!
1000 added!
1000 added!
1000 added!
Shared data = 1000
ramon@ramon-silva:~/Downloads/so$
```

5. Dê uma descrição do que faz cada um desses tipos ou funções.
6. Rode o programa. O que ele faz? Explique cada linha do resultado da saída do programa.
7. Por que as threads do programa não terminam na ordem em que são criadas?
9. O que a função `sys.exit()` faz nesse caso?
10. Observe o programa `simpleMutexEx.py`. Para entender o que ele faz, procure na documentação [API da biblioteca de threads para Python3](#) os tipos e as funções que você não conhece:
 1. classe Lock
 2. Lock.acquire()
 3. Lock.release()
11. Dê uma descrição do que faz cada um desses tipos ou funções.
12. Rode o programa. O que ele faz?
13. Retire as diretivas de sincronização "`acquire()`" e "`release()`". O que acontece com o programa? Você saberia explicar esse comportamento?

Sem a diretivas *acquire()* e *release()* quando o programa chega na parte do "for" todos os threads tentam acessar a variável global ao "mesmo" tempo, fazendo com que a variável não consiga incrementar, as diretivas entram justamente nesse quesito pois elas fazem esse controle.