Python - Comunicação entre threads

Threads compartilham a memória alocada para um processo. Como resultado, threads no mesmo processo podem se comunicar entre si. Para facilitar a comunicação entre threads, o módulo de threading fornece o objeto Event e o objeto Condition.

O objeto de evento

Um objeto Event gerencia o estado de um sinalizador interno. O sinalizador é inicialmente falso e se torna verdadeiro com o método set() e redefinido para falso com o método clear(). O método wait() é bloqueado até que o sinalizador seja verdadeiro.

Métodos do objeto Evento -

método is_set()

Retorne True se e somente se o sinalizador interno for verdadeiro.

método set()

Defina o sinalizador interno como verdadeiro. Todos os threads que esperam que isso se torne realidade são despertados. Threads que chamam wait() quando o sinalizador é verdadeiro não serão bloqueados.

método claro()

Redefina o sinalizador interno para falso. Posteriormente, os threads que chamam wait() serão bloqueados até que set() seja chamado para definir o sinalizador interno como verdadeiro novamente.

método wait(timeout=None)

Bloqueie até que o sinalizador interno seja verdadeiro. Se o sinalizador interno for verdadeiro na entrada, retorne imediatamente. Caso contrário, bloqueie até que outro thread chame set() para definir o sinalizador como verdadeiro ou até que ocorra o tempo limite opcional.

Quando o argumento timeout estiver presente e não None, deverá ser um número de ponto flutuante especificando um tempo limite para a operação em segundos.

Exemplo

O código a seguir tenta simular o fluxo de tráfego sendo controlado pelo estado do sem VERDE ou VERMELHO.

Existem dois threads no programa, visando duas funções diferentes. A função signal_state() ativa e zera periodicamente o evento indicando mudança de sinal de VERDE para VERMELHO.

A função tráfego_flow() espera que o evento seja definido e executa um loop até que ele permaneça definido.

```
from threading import *
import time

def signal_state():
   while True:
      time.sleep(5)
      print("Traffic Police Giving GREEN Signal")
      event.set()
      time.sleep(10)
      print("Traffic Police Giving RED Signal")
      event.clear()

def traffic_flow():
   num=0
   while num<10:
      print("Waiting for GREEN Signal")
      event.wait()
      print("GREEN Signal ... Traffic can move")
      while event.is_set():
         num=num+1
         print("Vehicle No:", num," Crossing the Signal")
         time.sleep(2)
      print("RED Signal ... Traffic has to wait")

event=Event()
t1=Thread(target=signal_state)
t2=Thread(target=traffic_flow)
t1.start()
t2.start()
```

Saída

```
Waiting for GREEN Signal
Traffic Police Giving GREEN Signal
GREEN Signal ... Traffic can move
Vehicle No: 1 Crossing the Signal
Vehicle No: 2 Crossing the Signal
Vehicle No: 3 Crossing the Signal
```

```
Vehicle No: 4 Crossing the Signal
Vehicle No: 5 Crossing the Signal
Signal is RED
RED Signal ... Traffic has to wait
Waiting for GREEN Signal
Traffic Police Giving GREEN Signal
GREEN Signal ... Traffic can move
Vehicle No: 6 Crossing the Signal
Vehicle No: 7 Crossing the Signal
Vehicle No: 8 Crossing the Signal
Vehicle No: 9 Crossing the Signal
Vehicle No: 10 Crossing the Signal
```

O objeto de condição

A classe de condição na classe do módulo de threading implementa objetos de variável de condição. O objeto de condição força um ou mais threads a esperar até serem notificados por outro thread. A condição está associada a um bloqueio reentrante. Um objeto de condição possui métodos adquire() e release() que chamam os métodos correspondentes do bloqueio associado.

```
threading.Condition(lock=None)
```

A seguir estão os métodos do objeto Condition -

adquirir(*args)

Adquira o bloqueio subjacente. Este método chama o método correspondente no bloqueio subjacente; o valor de retorno é o que o método retorna.

liberar()

Libere o bloqueio subjacente. Este método chama o método correspondente no bloqueio subjacente; Não há reembolso.

espere(tempo limite=Nenhum)

Este método libera o bloqueio subjacente e, em seguida, bloqueia até que ele seja despertado por uma chamada notify() ou notify_all() para a mesma variável de condição em outro thread, ou até que o tempo limite opcional ocorra. Uma vez despertado ou expirado, ele readquire o bloqueio e retorna.

wait_for(predicado, tempo limite=Nenhum)

Este método utilitário pode chamar wait() repetidamente até que o predicado seja satisfeito ou até que ocorra um tempo limite. O valor de retorno é o último valor de retorno do predicado e será avaliado como False se o tempo limite do método expirar.

notificar(n=1)

Este método acorda no máximo n threads aguardando a variável de condição; é autônomo se nenhum thread estiver esperando.

notificar_todos()

Acorde todos os threads aguardando nesta condição. Este método funciona como notify(), mas ativa todos os threads em espera em vez de um. Se o thread de chamada não tiver adquirido o bloqueio quando esse método for chamado, um RuntimeError será gerado.

Exemplo

No código a seguir, o thread t2 executa a função taskB() e t1 executa a função taskA(). O thread t1 adquire a condição e a notifica. Nesse momento, o thread t2 está em estado de espera. Depois que a condição é liberada, o thread em espera consome o número aleatório gerado pela função de notificação.

```
from threading import *
import time
import random

numbers=[]
def taskA(c):
   while True:
      c.acquire()
      num=random.randint(1,10)
      print("Generated random number:", num)
      numbers.append(num)
      print("Notification issued")
      c.notify()
      c.release()
      time.sleep(5)

def taskB(c):
   while True:
      c.acquire()
      print("waiting for update")
      c.wait()
      print("Obtained random number", numbers.pop())
      c.release()
```

```
        time.sleep(5)

c=Condition()
t1=Thread(target=taskB, args=(c,))
t2=Thread(target=taskA, args=(c,))
t1.start()
t2.start()
```

Quando você executa este código, ele produzirá a seguinte **saída** -

```
waiting for update
Generated random number: 4
Notification issued
Obtained random number 4
waiting for update
Generated random number: 6
Notification issued
Obtained random number 6
waiting for update
Generated random number: 10
Notification issued
Obtained random number 10
waiting for update
```