Python - Agendamento de Threads

Python oferece suporte a vários threads em um programa. Um programa multithread pode executar múltiplas subtarefas de forma independente, o que permite a execução paralela de tarefas.

O intérprete Python mapeia solicitações de thread Python para POSIX/pthreads ou threads do Windows. Conseqüentemente, semelhante aos threads comuns, os threads Python são manipulados pelo sistema operacional host.

No entanto, não há suporte para agendamento de threads no interpretador Python. Conseqüentemente, prioridade de thread, esquemas de agendamento e preempção de thread não são possíveis com o interpretador Python. O agendamento e a troca de contexto dos threads Python estão à disposição do agendador do host.

Python tem algum suporte para agendamento de tarefas na forma de módulo sched como biblioteca padrão. Pode ser utilizado na criação de bots e outras aplicações de monitoramento e automação. O módulo **sched** implementa um agendador de eventos genérico para executar tarefas em horários específicos. Ele fornece ferramentas semelhantes, como agendador de tarefas no Windows ou Linux.

A classe do agendador é definida no módulo integrado **sched .**

```
scheduler(timefunc=time.monotonic, delayfunc=time.sleep)
```

Os métodos definidos na classe do agendador incluem -

- **agendador.enter()** - Os eventos podem ser agendados para execução após um atraso ou em um horário específico. Para agendá-los com atraso, o método enter() é usado.
- **agendador.cancel()** - Remova o evento da fila. Se o evento não for um evento atualmente na fila, este método gerará um ValueError.
- **agendador.run(blocking=True)** - Execute todos os eventos agendados.

Os eventos podem ser programados para serem executados após um atraso ou em um horário específico. Para agendá-los com atraso, use o método enter(), que leva quatro argumentos.

- Um número que representa o atraso
- Um valor prioritário
- A função a ser chamada

- Uma tupla de argumentos para a função

Exemplo 1

Este exemplo agenda dois eventos diferentes -

```
import sched
import time

scheduler = sched.scheduler(time.time, time.sleep)

def schedule_event(name, start):
   now = time.time()
   elapsed = int(now - start)
   print('elapsed=',elapsed, 'name=', name)

start = time.time()
print('START:', time.ctime(start))
scheduler.enter(2, 1, schedule_event, ('EVENT_1', start))
scheduler.enter(5, 1, schedule_event, ('EVENT_2', start))

scheduler.run()
```

Ele produzirá a seguinte **saída** -

```
START: Mon Jun 5 15:37:29 2023
elapsed= 2 name= EVENT_1
elapsed= 5 name= EVENT_2
```

Exemplo 2

Vamos dar outro exemplo para entender melhor o conceito -

```
import sched
from datetime import datetime
import time

def addition(a,b):
   print("Performing Addition : ", datetime.now())
   print("Time : ", time.monotonic())
   print("Result : ", a+b)

s = sched.scheduler()
```

```
print("Start Time : ", datetime.now())

event1 = s.enter(10, 1, addition, argument = (5,6))
print("Event Created : ", event1)
s.run()
print("End Time : ", datetime.now())
```

Ele produzirá a seguinte **saída** -

```
Start Time : 2023-06-05 15:49:49.508400
Event Created : Event(time=774087.453, priority=1, sequence=0, action=<function addition at 0x
Performing Addition : 2023-06-05 15:49:59.512213
Time : 774087.484
Result : 11
End Time : 2023-06-05 15:49:59.559659
```