Barrier

This lesson discusses the all-important barrier synchronization construct.

Barrier

A barrier is a synchronization construct to wait for a certain number of threads to reach a common synchronization point in code. The involved threads each invoke the barrier object's **`wait()`** method and get blocked till all of threads have called **`wait()`**. When the last thread invokes **`wait()`** all of the waiting threads are released simultaneously. The below snippet shows example usage of barrier:

Using a barrier

```
from threading import Barrier
from threading import Thread
import random
import time


def thread_task():
    time.sleep(random.randint(0, 7))
    print("\nCurrently {0} threads blocked on barrier".format(
barrier.n_waiting))
    barrier.wait()


num_threads = 5
barrier = Barrier(num_threads)
threads = [0] * num_threads

for i in range(num_threads):
    threads[i - 1] = Thread(target=thread_task)

for i in range(num_threads):
    threads[i].start()
```

```
from threading import Barrier
from threading import Thread
import random
import time


def thread_task():
    time.sleep(random.randint(0, 7))
    print("\nCurrently {0} threads blocked on barrier".format(barrier.n_waiting))
    barrier.wait()


num_threads = 5
barrier = Barrier(num_threads)
threads = [0] * num_threads

for i in range(num_threads):
    threads[i - 1] = Thread(target=thread_task)

for i in range(num_threads):
    threads[i].start()
```

The barrier constructor also accepts a callable argument as an action to be performed when threads are released. Only one of the threads released will invoke the action. An example is given below:

```
from threading import Barrier
from threading import Thread
from threading import current_thread
import random
import time


def thread_task():
    time.sleep(random.randint(0, 5))
    print("\nCurrently {0} threads blocked on barrier".format(
barrier.n_waiting))
    barrier.wait()


def when_all_threads_released():
    print("All threads released, reported by {0}".format(curre
nt_thread().getName()))


num_threads = 5
barrier = Barrier(num_threads, action=when_all_threads_release
d)
threads = [0] * num_threads

for i in range(num_threads):
    threads[i - 1] = Thread(target=thread_task)

for i in range(num_threads):
    threads[i].start()
```

```
from threading import Barrier
from threading import Thread
from threading import current_thread
import random
import time


def thread_task():
    time.sleep(random.randint(0, 5))
    print("\nCurrently {0} threads blocked on barrier".format(barrier.n_waiting))
    barrier.wait()


def when_all_threads_released():
    print("All threads released, reported by {0}".format(current_thread().getName()


num_threads = 5
barrier = Barrier(num_threads, action=when_all_threads_released)
threads = [0] * num_threads

for i in range(num_threads):
    threads[i - 1] = Thread(target=thread_task)

for i in range(num_threads):
    threads[i].start()
```

If you execute the above snippet multiple times, you'll see that the action passed into barrier constructor is executed by a randomly chosen thread each time.

Broken Barriers

The barrier object exposes an **`abort()`** method which can be invoked to avoid deadlocks if needed. Threads already waiting on a barrier experience a **`BrokenBarrierError`** if **`abort()`** is invoked. The example below demonstrates this scenario.

Breaking barriers

```
from threading import Barrier
from threading import Thread
import time


def thread_task():
    print("\nCurrently {0} threads blocked on barrier".format(
barrier.n_waiting))
    barrier.wait()
    print("Barrier broken")


num_threads = 5
barrier = Barrier(num_threads + 1)
threads = [0] * num_threads

for i in range(num_threads):
    threads[i - 1] = Thread(target=thread_task)

for i in range(num_threads):
    threads[i].start()

time.sleep(3)

print("Main thread about to invoke abort on barrier")
barrier.abort()
```

```
from threading import Barrier
from threading import Thread
import time


def thread_task():
    print("\nCurrently {0} threads blocked on barrier".format(barrier.n_waiting))
    barrier.wait()
    print("Barrier broken")


num_threads = 5
barrier = Barrier(num_threads + 1)
threads = [0] * num_threads

for i in range(num_threads):
    threads[i - 1] = Thread(target=thread_task)

for i in range(num_threads):
    threads[i].start()

time.sleep(3)

print("Main thread about to invoke abort on barrier")
barrier.abort()
```

Back

Next

Timer

With

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/barrier__threading-module__python-concurrency-for-senior-engineering-interviews)