Miscellaneous Functions

This lesson demonstrates the use of various utility functions present in the concurrent module.

Miscellaneous Functions

The **`concurrent.futures`** module provides two methods to wait for futures collectively. These are:

- wait
- as_completed

Let's discuss each with examples below.

wait ( )

The **`wait()`** method takes in an iterable consisting of futures. These futures could be futures returned from thread or process pools. Next, it takes a timeout value, which if elapsed will return from the **`wait()`** call irrespective of how many futures have completed. The last argument is interesting and describes the strategy used to wait for the futures. It takes on the following values:

- FIRST_COMPLETED
- FIRST_EXCEPTION

- ALL_COMPLETED

The **`wait()`** call returns a value consisting of two sets. One set consists of futures that have completed or cancelled and the other consists of futures pending or running futures. Consider the snippet from the executable example below:

```
    result = wait(lst, timeout=None, return_when='ALL_COMPLETE
D')

    print("completed futures count: " + str(len(result.done))
 + " and uncompleted futures count: " +
          str(len(result.not_done)) + "\n")
```

The above lines demonstrate how we **`wait()`** for all futures to complete and then print the counts of both the sets returned from the **`wait()`** call.

```
from concurrent.futures import wait
from concurrent.futures import ThreadPoolExecutor
from concurrent.futures import ProcessPoolExecutor


def square(item):
    return item * item


if __name__ == '__main__':
    lst = list()
    threadExecutor = ThreadPoolExecutor(max_workers=10)
    processExecutor = ProcessPoolExecutor(max_workers=10)

    for i in range(1, 6):
        lst.append(threadExecutor.submit(square, i))

    for i in range(6, 11):
        lst.append(processExecutor.submit(square, i))

    result = wait(lst, timeout=None, return_when='ALL_COMPLETED')

    print("completed futures count: " + str(len(result.done)) + " and uncompleted f
          str(len(result.not_done)) + "\n")

    for ftr in result.done:
        print(ftr.result())

    threadExecutor.shutdown()
    processExecutor.shutdown()
```

Next we'll tweak our example to change the **return_when** argument to FIRST_COMPLETED and introduce a sleep for all tasks except one and print the counts for the two sets returned.

```
from concurrent.futures import wait
from concurrent.futures import ThreadPoolExecutor
from concurrent.futures import ProcessPoolExecutor
import time

def square(item):
    if item != 1:
        time.sleep(4)

    return item * item


if __name__ == '__main__':
    lst = list()
    threadExecutor = ThreadPoolExecutor(max_workers=10)
    processExecutor = ProcessPoolExecutor(max_workers=10)

    for i in range(1, 6):
        lst.append(threadExecutor.submit(square, i))

    for i in range(6, 11):
        lst.append(processExecutor.submit(square, i))

    result = wait(lst, timeout=0.01, return_when='FIRST_COMPLETED')
    print("completed futures count: " + str(len(result.done)) + " and uncompleted f
          str(len(result.not_done)) + "\n")

    threadExecutor.shutdown()
    processExecutor.shutdown()
```

An interesting observation about the above code is that we changed the **timeout** to 0.01 second. The `wait()` call would return after timeout has elapsed even though nine of the futures haven't completed. In the example below we make all the tasks sleep for four seconds and the count of not_done set becomes ten.

```
from concurrent.futures import wait
from concurrent.futures import ThreadPoolExecutor
from concurrent.futures import ProcessPoolExecutor
import time

def square(item):
    time.sleep(4)
    return item * item


if __name__ == '__main__':
    lst = list()
    threadExecutor = ThreadPoolExecutor(max_workers=10)
    processExecutor = ProcessPoolExecutor(max_workers=10)

    for i in range(1, 6):
        lst.append(threadExecutor.submit(square, i))

    for i in range(6, 11):
        lst.append(processExecutor.submit(square, i))

    result = wait(lst, timeout=0.01, return_when='FIRST_COMPLETED')
    print("completed futures count: " + str(len(result.done)) + " and uncompleted f
          str(len(result.not_done)) + "\n")

    threadExecutor.shutdown()
    processExecutor.shutdown()
```

as_completed ( )

The **`as_completed()`** method takes in an iterable of futures returned from either a thread or a process pool. It returns an iterator over the future instances. The iterator returns futures as they complete. However, if **timeout** has elapsed since the **`as_completed()`** call and a future hasn't completed, then a timeout exception is raised.

In the example below, we sleep each task in such a way that the highest valued argument to the **`square()`** method sleeps for zero seconds and the least valued argument sleeps for one second. The iterator ends up printing the squares in descending order.

```
from concurrent.futures import as_completed
from concurrent.futures import ProcessPoolExecutor
import time


def square(item):
    time.sleep(5 - item)
    return item * item


if __name__ == '__main__':
    lst = list()
    processExecutor = ProcessPoolExecutor(max_workers=10)

    for i in range(1, 6):
        lst.append(processExecutor.submit(square, i))

    result = as_completed(lst, timeout=None)

    for ftr in result:
        print(ftr.result())

    processExecutor.shutdown()
```

We tweak the above example to make one of the tasks sleep for three seconds and set the timeout in **`as_completed()`** to one second. The timeout is long enough for the other four tasks to complete but the fifth one doesn't complete when we iterate over it and a timeout exception is raised. The example below demonstrates this setup.

```
from concurrent.futures import as_completed
from concurrent.futures import ProcessPoolExecutor
import time


def square(item):
    if item == 5:
      time.sleep(3)
    return item * item


if __name__ == '__main__':
    lst = list()
    processExecutor = ProcessPoolExecutor(max_workers=10)

    for i in range(1, 6):
        lst.append(processExecutor.submit(square, i))

    result = as_completed(lst, timeout=1)

    for ftr in result:
        print(ftr.result())

    processExecutor.shutdown()
```

Another interesting observation is if we set the **timeout** to be extremely small then none of the futures complete as exhibited below.

```
from concurrent.futures import as_completed
from concurrent.futures import ProcessPoolExecutor
import time


def square(item):
    return item * item


if __name__ == '__main__':
    lst = list()
    processExecutor = ProcessPoolExecutor(max_workers=10)

    for i in range(1, 6):
        lst.append(processExecutor.submit(square, i))

    result = as_completed(lst, timeout=0.00001)

    for ftr in result:
        print(ftr.result())

    processExecutor.shutdown()
```

← Back
Futures

Next →
Introduction

✓ Mark as Completed

Report an Issue

Ask a Question
(https://discuss.educative.io/tag/miscellaneous-functions__concurrent-package__python-concurrency-for-senior-engineering-interviews)