assignment=11

July 29, 2023

1 Q1. What is multiprocessing in python? Why is it useful?

Multiprocessing in Python is a built-in package that allows the system to run multiple processes simultaneously. It will enable the breaking of applications into smaller threads that can run independently. The operating system can then allocate all these threads or processes to the processor to run them parallelly, thus improving the overall performance and efficiency.

`Why Use Multiprocessing In Python?`

Performing multiple operations for a single processor becomes challenging. As the number of processes keeps increasing, the processor will have to halt the current process and move to the next, to keep them going. Thus, it will have to interrupt each task, thereby hampering the performance.

You can think of it as an employee in an organization tasked to perform jobs in multiple departments. If the employee has to manage the sales, accounts, and even the backend, he will have to stop sales when he is into accounts and vice versa.

Suppose there are different employees, each to perform a specific task. It becomes simpler, right? That's why multiprocessing in Python becomes essential. The smaller task threads act like different employees, making it easier to handle and manage various processes. A multiprocessing system can be represented as:

A system with more than a single central processor A multi-core processor, i.e., a single computing unit with multiple independent core processing units In multiprocessing, the system can divide and assign tasks to different processors.

2 Q2. What are the differences between multiprocessing and multithreading?

What is Multiprocessing? Multiprocessing refers to using multiple CPUs/processors in a single system. Multiple CPUs can act in a parallel fashion and execute multiple processes together. A task is divided into multiple processes that run on multiple processors. When the task is over, the results from all processors are compiled together to provide the final output. Multiprocessing increases the computing power to a great extent. Symmetric multiprocessing and asymmetric multiprocessing are two types of multiprocessing.

Multiprocessing increases the reliability of the system because of the use of multiple CPUs. However, a significant amount of time and specific resources are required for multiprocessing. Multiprocessing is relatively more cost effective as compared to a single CPU system.

What is Multithreading? Multithreading refers to multiple threads being executed by a single CPU in such a way that each thread is executed in parallel fashion and CPU/processor is switched between them using context switch. Multithreading is a technique to increase the throughput of a processor. In multithreading, accessing memory addresses is easy because all of the threads share the same parent process.

The switching among different threads is fast and efficient. It must be noted that in multithreading, multiple threads are created from a single process and then these threads are processed simultaneously.

Difference between Multiprocessing and Multithreading The following table highlights the major differences between Multiprocessing and Multithreading –

Factor:	Multiprocessing	Multithreading
Concept :	Multiple processors/CPUs are added to the system to increase the comuting power of the system.	* multiple threades are crated of process to be executed in a a parallel to increase the throughtput of the system.
Parallel Action	Multiple processes are executed in a parallel fashion.	* Multiple threads are executed in a parallel fashion.
Categories:	Multiprocessing can be classified into symmetric and asymmetric multiprocessing.	* No such classification present for multithreading.
Time:	Process creation is time-consuming.	* Thread creation is easy and is time savvy.
Execution:	In multiprocessing, many processes are executed simultaneously.	* In multithreading, many threads are executed simultaneously.
Address space:	In multiprocessing, a separate address space is created for each process.	* In multithreading, a common address spece is used for all the threads.

```
  Resources: Multiprocessing requires a significant amount
           of time and large number of resources.      * Multithreading requires less timen
                                                        and few resources to create.
```

```
   Conclusion
The most significant difference between multiprocessing and multithreading is that
multiprocessing executes many processes at the same time, whereas multithreading
executes many threads of a process at the same time.
```

3 Q3. Write a python code to create a process using the multiprocessing module.

[11]:
```python
from multiprocessing import Process
def cube(x):
   for x in my_numbers:
    print('%s cube is %s' % (x, x**3))
def evenno(x):
   for x in my_numbers:
    if x % 2 == 0:
        print('%s is an even number ' % (x))
   if __name__ == '__main__':
      my_numbers = [3, 4, 5, 6, 7, 8]
      my_process1 = Process(target=cube, args=('x',))
      my_process2 = Process(target=evenno, args=('x',))
      my_process1.start()
      my_process2.start()
      my_process1.join()
   my_process2.join()
print ("Done")
```

```
Done
```

[68]:
```python
from multiprocessing import Process, Pipe
def myfunction(conn):
    conn.send(['hi!! I am Python'])
    conn.close()
    if __name__ == '__main__':
        parent_conn, child_conn = Pipe()
        p = Process(target=myfunction, args=(child_conn,))
        p.start()
    print (parent_conn.recv() )
    p.join()
```

[78]:
```python
from multiprocessing import Process
def cube(x):
        for x in my_numbers:
            print('%s cube is %s' % (x, x**3))
        if __name__ == '__main__':
            my_numbers = [3, 4, 5, 6, 7, 8]
            p = Process(target=cube, args=('x',))
            p.start()
            p.join
print ("Done")
```

```
Done
```

[104]:
```
 ['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
```

[104]:
```
['__doc__',
 '__file__',
 '__name__',
 'acos',
 'asin',
 'atan',
 'atan2',
 'ceil',
 'cos',
 'cosh',
 'degrees',
 'e',
 'exp',
 'fabs',
 'floor',
 'fmod',
 'frexp',
 'hypot',
 'ldexp',
 'log',
 'log10',
 'modf',
 'pi',
 'pow',
 'radians',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh']
```

[122]:
```
import multiprocessing
def evenno(numbers, q):
    for n in numbers:
        if n % 2 == 0:
            q.put(n)
        if __name__ == "__main__":
            q = multiprocessing.Queue()
            p = multiprocessing.Process(target=evenno, args=(range(10), q))
            p.start()
            p.join()
            while q:
                print(q.get())
```

4 Q4. What is a multiprocessing pool in python? Why is it used?

In this tutorial you will discover the difference between the multiprocessing pool and multiprocessing.Process and when to use each in your Python projects.

Let's get started.

Skip the tutorial. Master the multiprocessing Pool today. Learn how

Table of Contents What is a multiprocessing.Pool What is a multiprocessing.Process Execute a Target Function Extend the Class Comparison of Pool vs Process Similarities Between Pool and Process Differences Between Pool and Process Summary of Differences How to Choose Pool or Process When to Use multiprocessing.Pool Don't Use multiprocessing.Pool When… When to Use the multiprocessing.Process Don't Use multiprocessing.Process When… Further Reading Takeaways What is a multiprocessing.Pool The multiprocessing.pool.Pool class provides a process pool in Python.

Note, you can access the process pool class via the helpful alias multiprocessing.Pool.

It allows tasks to be submitted as functions to the process pool to be executed concurrently.

A process pool object which controls a pool of worker processes to which jobs can be submitted. It supports asynchronous results with timeouts and callbacks and has a parallel map implementation.

— MULTIPROCESSING — PROCESS-BASED PARALLELISM A process pool is a programming pattern for automatically managing a pool of worker processes.

The pool can provide a generic interface for executing ad hoc tasks with a variable number of arguments, much like the target property on the Process object, but does not require that we choose a process to run the task, start the process, or wait for the task to complete.

To use the process pool, we must first create and configure an instance of the class.

create a process pool pool = multiprocessing.pool.Pool(…) By default, the process pool will have one worker process for each logical CPU core in your system.

We can specify the number of workers to create via an argument to the class constructor.

create a process pool pool = multiprocessing.pool.Pool(4) Tasks are issued in the process pool by specifying a function to execute that may or may not have arguments and may or may not return a value.

We can issue one-off tasks to the process pool using functions such as apply() or we can apply the same function to an iterable of items using functions such as map().

*The Pool class in multiprocessing can handle an enormous number of processes. It allows you to run multiple jobs per process (due to its ability to queue the jobs). The memory is allocated only to the executing processes, unlike the Process class, which allocates memory to all the processes.

*Use the multiprocessing. Pool class when you need to execute tasks that may or may not take arguments and may or may not return a result once the tasks are complete. Use the multiprocessing. Pool class when you need to execute different types of ad hoc tasks, such as calling different target task functions.

5 Q5. How can we create a pool of worker processes in python using the multiprocessing module?

from multiprocessing. pool import Pool if **name** == '**main**': # create a process pool with the default number of workers. pool = Pool() # report the status of the process pool. print(pool) # report the number of processes in the pool. print(pool. _processes)

children = active_children() : print(len(children))

The multiprocessing.pool.Pool in Python provides a pool of reusable processes for executing ad hoc tasks.

A process pool can be configured when it is created, which will prepare the child workers.

We can issue one-off tasks to the process pool using functions such as apply() or we can apply the same function to an iterable of items using functions such as map(). Results for issued tasks can then be retrieved synchronously, or we can retrieve the result of tasks later by using asynchronous versions of the functions such as apply_async() and map_async().

The process pool has a fixed number of worker processes.

It is important to limit the number of worker processes in the process pools to perhaps the number of logical CPU cores or the number of physical CPU cores in your system, depending on the types of tasks we will be executing.

[156]:
```python
# SuperFastPython.com
# example of configuring the number of worker processes
from multiprocessing.pool import Pool

# protect the entry point
if __name__ == '__main__':
    # create a process pool with many workers
    with Pool(60) as pool:
        # report the status of the pool
        print(pool)
```

```
<multiprocessing.pool.Pool state=RUN pool_size=60>
```

6 Q6. Write a python program to create 4 processes, each process should print a different number using the multiprocessing module in python.

[157]:
```
from multiprocessing import Pool
def f(x):
    return x*x
with Pool(5) as p:
    print(p.map(f,[1,2,3]))
```

```
[1, 4, 9]
```

[158]:
```
>>> import multiprocessing
>>> multiprocessing.cpu_count()
```

[158]:
```
64
```

[159]:
```
import multiprocessing
from multiprocessing import Process
def testing():
      print("Works")
def square(n):
       print("The number squares to ",n**2)
def cube(n):
       print("The number cubes to ",n**3)
if __name__=="__main__":
   p1=Process(target=square,args=(7,))
   p2=Process(target=cube,args=(7,))
   p3=Process(target=testing)
   p1.start()
   p2.start()
   p3.start()
   p1.join()
   p2.join()
   p3.join()
   print("We're done")
```

```
The number squares to  49
The number cubes to  343Works

We're done
```

[160]:
```
import multiprocessing
from multiprocessing import Process
import os
def child1():
     print("Child 1",os.getpid())
```

```
def child2():
        print("Child 2",os.getpid())
if __name__=="__main__":
   print("Parent ID",os.getpid())
   p1=Process(target=child1)
   p2=Process(target=child2)
   p1.start()
   p2.start()
   p1.join()
   alive='Yes' if p1.is_alive() else 'No'
   print("Is p1 alive?",alive)
   alive='Yes' if p2.is_alive() else 'No'
   print("Is p2 alive?",alive)
   p2.join()
   print("We're done")
```

```
Parent ID 77
Child 1 3456Child 2
 3459
Is p1 alive? No
Is p2 alive? Yes
We're done
```

[161]:
```
import multiprocessing
from multiprocessing import Process, current_process
import os
def child1():
     print(current_process().name)
def child2():
         print(current_process().name)
if __name__=="__main__":
   print("Parent ID",os.getpid())
   p1=Process(target=child1,name='Child 1')
   p2=Process(target=child2,name='Child 2')
   p1.start()
   p2.start()
   p1.join()
   p2.join()
   print("We're done")
```

```
Parent ID 77
Child 1
Child 2
We're done
```

[162]:
```
from multiprocessing import Process, Lock
lock=Lock()
```

```
def printer(item):
  lock.acquire()
  try:
      print(item)
  finally:
      lock.release()
if __name__=="__main__":
  items=['nacho','salsa',7]
  for item in items:
     p=Process(target=printer,args=(item,))
     p.start()
```

```
nacho
salsa
7
```

[163]:
```
from multiprocessing import Pool
def double(n):
   return n*2
if __name__=='__main__':
   nums=[2,3,6]
   pool=Pool(processes=3)
   print(pool.map(double,nums))
```

```
[4, 6, 12]
```

[166]:
```
from multiprocessing import Pool
def double(n):
   return n*2
if __name__=='__main__':
   pool=Pool(processes=3)
   result=pool.apply_async(double,(7,))
   print(result.get(timeout=1))
```

```
14
```

[ ]:

[ ]:

[ ]:

[ ]: