Multiprocessing_Assignment

August 22, 2023

[7]:
```
'''Q1. What is multiprocessing in Python? Why is it useful?
Multiprocessing is a module in Python that allows you to create and manage␣
 ↪multiple processes. Each process runs in its own separate memory space,␣
 ↪allowing for true parallel execution on multi-core CPUs. This is useful for␣
 ↪achieving better performance and improved utilization of available␣
 ↪resources, especially for CPU-bound tasks.
'''
```

[7]:
```
'Q1. What is multiprocessing in Python? Why is it useful?\nMultiprocessing is a
module in Python that allows you to create and manage multiple processes. Each
process runs in its own separate memory space, allowing for true parallel
execution on multi-core CPUs. This is useful for achieving better performance
and improved utilization of available resources, especially for CPU-bound
tasks.\n'
```

[6]:
```
'''Q-2. What are the differences between multiprocessing and multithreading?

1. In multiprocessing, multiple processes run in separate memory spaces and can␣
 ↪achieve true parallelism, while in multithreading, multiple threads      ␣
 ↪share the same memory space and are limited by the Global Interpreter Lock␣
 ↪(GIL) in CPython, allowing only one thread to execute Python bytecode at    ␣
 ↪a time.
2. Multiprocessing is better suited for CPU-bound tasks, while multithreading␣
 ↪is often used for I/O-bound tasks.
3. Multiprocessing has more memory overhead due to separate memory spaces,␣
 ↪while multithreading has less memory overhead.
4. Multiprocessing can fully utilize multiple CPU cores, whereas multithreading␣
 ↪may not achieve true parallelism due to the GIL.'''
```

[6]:
```
'Q-2. What are the differences between multiprocessing and multithreading?\n\n1.
In multiprocessing, multiple processes run in separate memory spaces and can
achieve true parallelism, while in multithreading, multiple threads      share
the same memory space and are limited by the Global Interpreter Lock (GIL) in
CPython, allowing only one thread to execute Python bytecode at    a time.\n2.
Multiprocessing is better suited for CPU-bound tasks, while multithreading is
often used for I/O-bound tasks.\n3. Multiprocessing has more memory overhead due
to separate memory spaces, while multithreading has less memory overhead.\n4.
```

```
Multiprocessing can fully utilize multiple CPU cores, whereas multithreading may
not achieve true parallelism due to the GIL.'
```

[5]:
```python
# Q-3:

import multiprocessing

def print_square(number):
    print("Square:", number * number)

if __name__ == "__main__":
    process = multiprocessing.Process(target=print_square, args=(5,))
    process.start()
    process.join()
```

```
Square: 25
```

Q-4:

A multiprocessing pool is a way to create a group of worker processes that can be used to parallelize and distribute the execution of a function across multiple inputs.

It is used to achieve parallelism when applying a function to a large set of data, improving efficiency and performance.

[3]:
```python
# Q-5:

import multiprocessing

def print_square(number):
    return number * number

if __name__ == "__main__":
    with multiprocessing.Pool(processes=4) as pool:
        results = pool.map(print_square, [1, 2, 3, 4])
        print(results)
```

```
[1, 4, 9, 16]
```

[4]:
```python
# Q-6:

import multiprocessing

def print_number(number):
    print("Number:", number)

if __name__ == "__main__":
    processes = []
    for i in range(1, 5):
```

```
        process = multiprocessing.Process(target=print_number, args=(i,))
        processes.append(process)
        process.start()

    for process in processes:
        process.join()
```

```
Number: 1
Number: 2
Number: 3
Number: 4
```

[ ]: