Multiprocessing in Python

Multiprocessing is a Python module that provides a simple way to run multiple processes in parallel. It allows you to take advantage of multiple cores or processors on your system and can significantly improve the performance of your code. In this repl, we'll take a closer look at the multiprocessing module and its various functions and how they can be used in Python.

Importing Multiprocessing

We can use multiprocessing by importing the multiprocessing module.

```
import multiprocessing
```

Now, to use multiprocessing we need to create a `process object` which calls a `start()` method. The start() method runs the process and then to stop the execution, we use the `join()` method. Here's how we can create a simple process.

Creating a process

```
import multiprocessing
def my_func():
  print("Hello from process", multiprocessing.current_process().name)
  process = multiprocessing.Process(target=my_func)
  process.start()
  process.join()
```

Functions

The following are some of the most commonly used functions in the multiprocessing module:

- `multiprocessing.Process(target, args)`: This function creates a new process that runs the target function with the specified arguments.
- `multiprocessing.Pool(processes)`: This function creates a pool of worker processes that can be used to parallelize the execution of a function across multiple input values.
- `multiprocessing.Queue()`: This function creates a queue that can be used to communicate data between processes.
- `multiprocessing.Lock()`: This function creates a lock that can be used to synchronize access to shared resources between processes.

Creating a pool of worker processes

Creating a pool of worker processes is a common approach to using multiprocessing in Python. The idea is to create a pool of worker processes and then assign tasks to them as needed. This allows you to take advantage of multiple CPU cores and process tasks in parallel.

```
from multiprocessing import Pool

def process_task(task):
```

```
    # Do some work here
    print("Task processed:", task)

if __name__ == '__main__':
    tasks = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    with Pool(processes=4) as pool:
        results = pool.map(process_task, tasks)
```

Using a queue to communicate between processes

When working with multiple processes, it is often necessary to pass data between them. One way to do this is by using a queue. A queue is a data structure that allows data to be inserted at one end and removed from the other end. In the context of multiprocessing, a queue can be used to pass data between processes.

```
def producer(queue):
    for i in range(10):
        queue.put(i)


def consumer(queue):
    while True:
        item = queue.get()
        print(item)


queue = multiprocessing.Queue()
p1 = multiprocessing.Process(target=producer, args=(queue,))
p2 = multiprocessing.Process(target=consumer, args=(queue,))
p1.start()
p2.start()
```

Using a lock to synchronize access to shared resources

When working with multiprocessing in python, locks can be used to synchronize access to shared resources among multiple processes. A lock is an object that acts as a semaphore, allowing only one process at a time to execute a critical section of code. The lock is released when the process finishes executing the critical section.

```
def increment(counter, lock):
    for i in range(10000):
        lock.acquire()
        counter.value += 1
        lock.release()

if __name__ == '__main__':
    counter = multiprocessing.Value('i', 0)
    lock = multiprocessing.Lock()
```

```
    p1 = multiprocessing.Process(target=increment, args=(counter, lock))
    p2 = multiprocessing.Process(target=increment, args=(counter, lock))

    p1.start()
    p2.start()

    p1.join()
    p2.join()

    print("Counter value:", counter.value)
```

Conclusion

As you can see, the multiprocessing module provides a simple and efficient way to run multiple processes in parallel. Whether you need to create a new process, run a function across multiple input values, communicate data between processes, or synchronize access to shared resources, the multiprocessing module has you covered.

In conclusion, the multiprocessing module is a powerful tool for parallelizing code in Python. Whether you are a beginner or an experienced Python developer, the multiprocessing module is an essential tool to have in your toolbox.