Python Code:

Python Copy code

```python
import multiprocessing
import time
```

- `multiprocessing`: This module lets us create **separate processes** (like independent CPUs).
- `time`: Just to simulate a "delay" so we can *see* the parallel execution happening.

Python Copy code

```python
def task1():
    print(f"Task 1 started by {multiprocessing.current_process().name}")
    for i in range(5):
        print(f"Task 1 - processing item {i}")
        time.sleep(0.5)
    print("Task 1 completed.")
```

- `task1` is a function that does some work.
- It prints a message saying it started.
- Then it **loops 5 times**, each time printing a "processing item" message.
- It **sleeps for 0.5 seconds** to simulate doing some work.
- After the loop, it prints "Task 1 completed".

Python Copy code

```python
def task2():
    print(f"Task 2 started by
{multiprocessing.current_process().name}
")
    for i in range(3):
        print(f"Task 2 - processing item
{i}")
        time.sleep(1)
    print("Task 2 completed.")
```

- `task2` is a different function.
- It works similarly but:
    - **Loops only 3 times** (less work).
    - **Sleeps for 1 second** (longer work per item).

Python Copy code

```python
if __name__ == "__main__":
```

- This checks if the Python file is being **run directly**, not imported by another script.
- This is **important** for `multiprocessing` to work properly in Windows/Linux.

Python Copy code

```python
p1 =
multiprocessing.Process(target=task1,
name="Processor-1")
p2 =
multiprocessing.Process(target=task2,
name="Processor-2")
```

- p1 and p2 are **Process** objects.
- Each one is set to run **task1** and **task2** respectively.
- We also **name** them ("Processor-1" and "Processor-2") for clarity in the print outputs.

Python Copy code

```python
p1.start()
p2.start()
```

- This **starts** both processes.
- Now **task1** and **task2** are running **at the same time** (parallel execution).

Python Copy code

```python
p1.join()
p2.join()
```

- `join()` tells the **main program** to **wait** for both processes to **finish** before continuing.
- It ensures that "Both tasks are completed" will only print **after** `task1` and `task2` are done.