Q1.

""" Multiprocessing in Python is a programming technique that involves the use of multiple processes to perform tasks concurrently.

- Parallelism
- Isolation
- Improved Stability
- I/O-Bound Tasks
- Exploiting Multiple Cores

Q2.

""" Multiprocessing :

- In multiprocessing, multiple processes are created, each with its own memory space and resources. Each process runs independently, and communication between processes typically involves inter-process communication

(IPC) mechanisms like pipes or queues.

- Multiprocessing allows true parallelism, as processes run in separate memory

spaces and can fully utilize multiple CPU cores.

- Processes can be more memory-intensive because each process has its own memory space.

Mutithreading :

- In multithreading, multiple threads are created within a single process. Threads share the same memory space and resources within the process and can communicate more easily.

They are lighter-weight than processes.

- Multithreading can achieve concurrency but may not fully utilize multiple CPU cores, especially in Python due to the Global Interpreter Lock (GIL).

- Threads within a process share memory space and are more memory-efficient.

"""

Q3.

In [1]:
```
import multiprocessing

def print_numbers():
    for i in range(1, 6):
        print(f"Number: {i}")

if __name__ == "__main__":
    process = multiprocessing.Process(target=print_numbers)

    process.start()

    process.join()

    print("Process has finished.")
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Process has finished.
```

Q4.

""" A multiprocessing pool in Python, often referred to as a "Pool," is a part of the multiprocessing module that provides a convenient way to parallelize the execution of a function over a collection of data. It allows you to distribute the workload across multiple processes in a controlled and efficient manner.

- Parallel Execution
- Multiple Workers
- Efficiency
- Simplified API
- Task Management

"""

Q5.

In [2]:
```
import multiprocessing

def worker_function(item):
    return item * 2

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

    num_processes = 4
    with multiprocessing.Pool(processes=num_processes) as pool:
        results = pool.map(worker_function, data)

    print("Results:", results)
```

```
Results: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Q6.

In [3]:
```
import multiprocessing

def print_number(number):
    print(f"Process {number}: {number}")

if __name__ == "__main__":
    numbers = [1, 2, 3, 4]
    with multiprocessing.Pool(processes=4) as pool:

        pool.map(print_number, numbers)
```

```
Process 1: 1Process 3: 3Process 2: 2Process 4: 4
```