Experiment_9_OS
Name: Veeransh Shah
Reg ID: 221070063

Aim:
Implementing shared memory and IPC using Python

Theory:

Inter-process communication (IPC) enables processes to communicate with each other by exchanging data or signals. This is crucial in multi-process applications where independent processes need to collaborate to perform complex tasks. Shared memory is one of the mechanisms for IPC, allowing multiple processes to access common memory spaces. This approach is efficient because it avoids the overhead of inter-process data transfers through messaging, which can involve complex protocols and data copying.

Python's multiprocessing module provides a way to create processes that can execute tasks concurrently, and it includes features for different IPC mechanisms, including:

- Pipes and Queues: For passing messages between processes.
- Shared Memory Objects: Using shared variables (like arrays) or server-based managed objects (like the list in the example).

In the given example, the Manager() from the multiprocessing module is used to create a shared list that both writer and reader processes access. This type of shared memory management helps demonstrate how data can be safely and efficiently shared between processes.

Code:

```
# Shared memory and IPC using python
import multiprocessing
import time

def writer(shared_list):
```

```
    """Function to write data to the shared list."""
    for i in range(5):
        shared_list.append(i)
        print(f"Writer added: {i}")
        time.sleep(1)

def reader(shared_list):
    """Function to read data from the shared list."""
    old_len = 0
    while old_len < 5:
        if len(shared_list) > old_len:
            print(f"Reader read: {shared_list[old_len]}")
            old_len += 1
        time.sleep(0.5)

if __name__ == '__main__':
    manager = multiprocessing.Manager()
    shared_list = manager.list()


    writer_process = multiprocessing.Process(target=writer,
args=(shared_list,))
    reader_process = multiprocessing.Process(target=reader,
args=(shared_list,))

    writer_process.start()
    reader_process.start()

    writer_process.join()
    reader_process.join()



    # Manager and Shared List: We initiate a Manager() that allows creating
a shared list. This list can be accessed and modified by different
processes.
    # Writer and Reader Functions: The writer function adds integers to the
shared list, and the reader function reads these integers. To simulate
asynchronous behavior, there are delays (sleep) added.
```

```
   # Processes: Two separate processes are created for writing and
reading. They operate on the same shared_list object managed by the
Manager.
```

Output:

```
veeransh@veeransh-XPS-9315:~/Desktop/Lab_work$ /usr/bin/python3 /home/veeransh/Desktop/Lab_work/OS/files/assn9.py
Writer added: 0
Reader read: 0
Writer added: 1
Reader read: 1
Writer added: 2
Reader read: 2
Writer added: 3
Reader read: 3
Writer added: 4
Reader read: 4
veeransh@veeransh-XPS-9315:~/Desktop/Lab_work$
```

Conclusion:

The provided code illustrates a fundamental application of shared memory and IPC in Python, using multiprocessing to manage concurrent processes and shared data structures. This setup mimics a real-world scenario where different parts of an application need to share results and status updates efficiently without unnecessary data copying or complex synchronization mechanisms.