Experiment : 7
Date : 02/06/21
Author : Bonnie Simon

IPC using Message Queue, Shared Memory & Pipes

AIM

To Implement Inter Process Communication using Message queues, shared memory and pipes in python.

THEORY

Multiprocessing refers to the ability of a system to support more than one processor at the same time. Applications in a multiprocessing system are broken to smaller routines that run independently. The operating system allocates these threads to the processors improving performance of the system.

In multiprocessing, two processes can communicate together using :

Shared memory

The shared memory IPC means that the processes read and write to a shared physical memory region. The OS is involved in establishing the shared memory channel between the processes, this is implemented by the virtual to physical translation.

PIPE

The most simple form of the message passing IPC is called pipes. Pipes are characterized by two endpoints so that only two processes can communicate at a time. There is no notion of the message in the pipe, and there will just be a stream of bytes that are pushed into the pipe from one process to another.

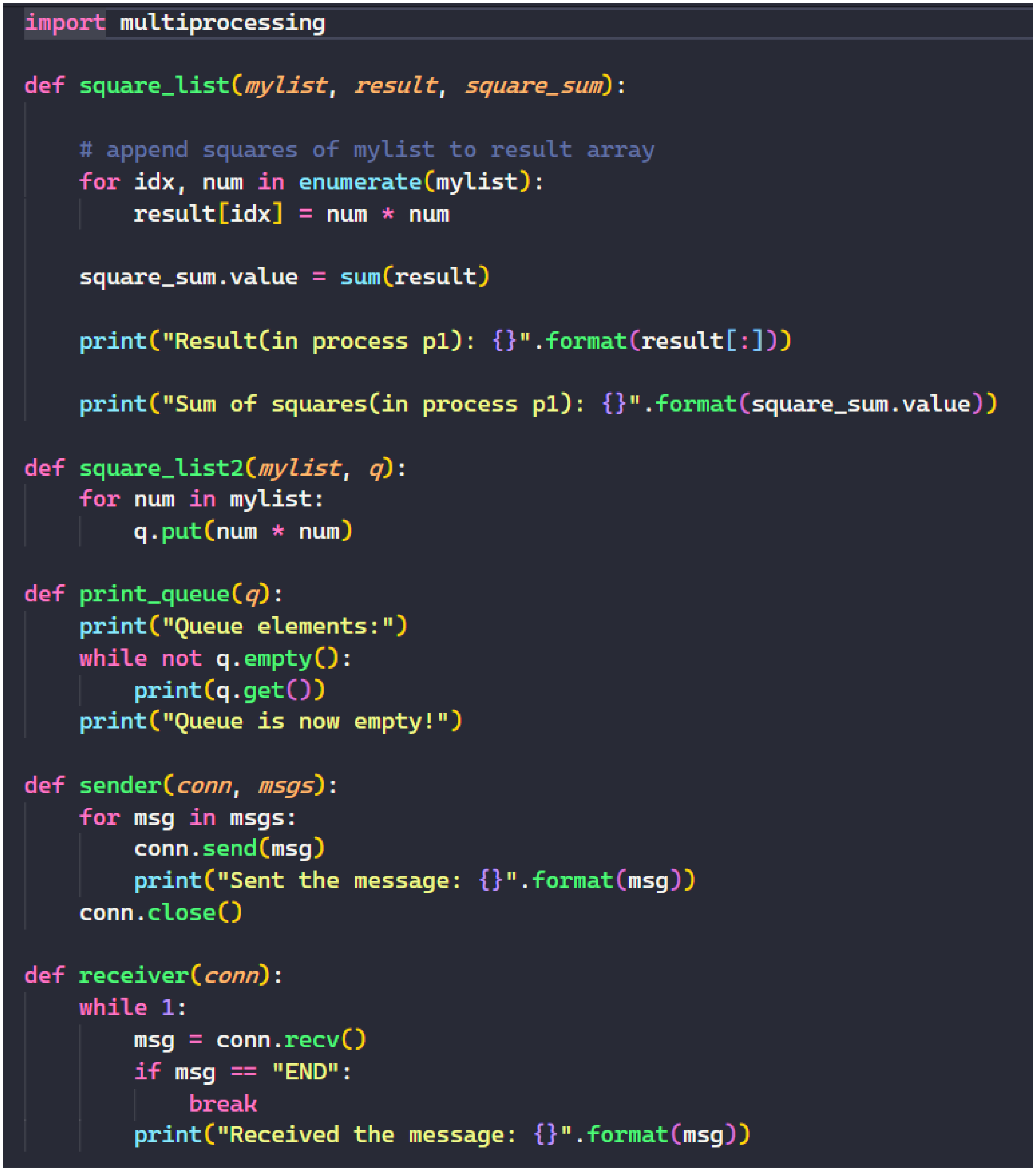
Message queue

A more complex form of message passing IPC is the message queue. As the name suggests, the message queues understand the notion of messages that they transfer. Thus, a sending process must submit a properly formatted message to the channel, and then the channel will deliver a properly formatted message to the receiving process. The OS level functionality regarding message queues also includes things like understanding priorities of messages or scheduling the way messages are being delivered.

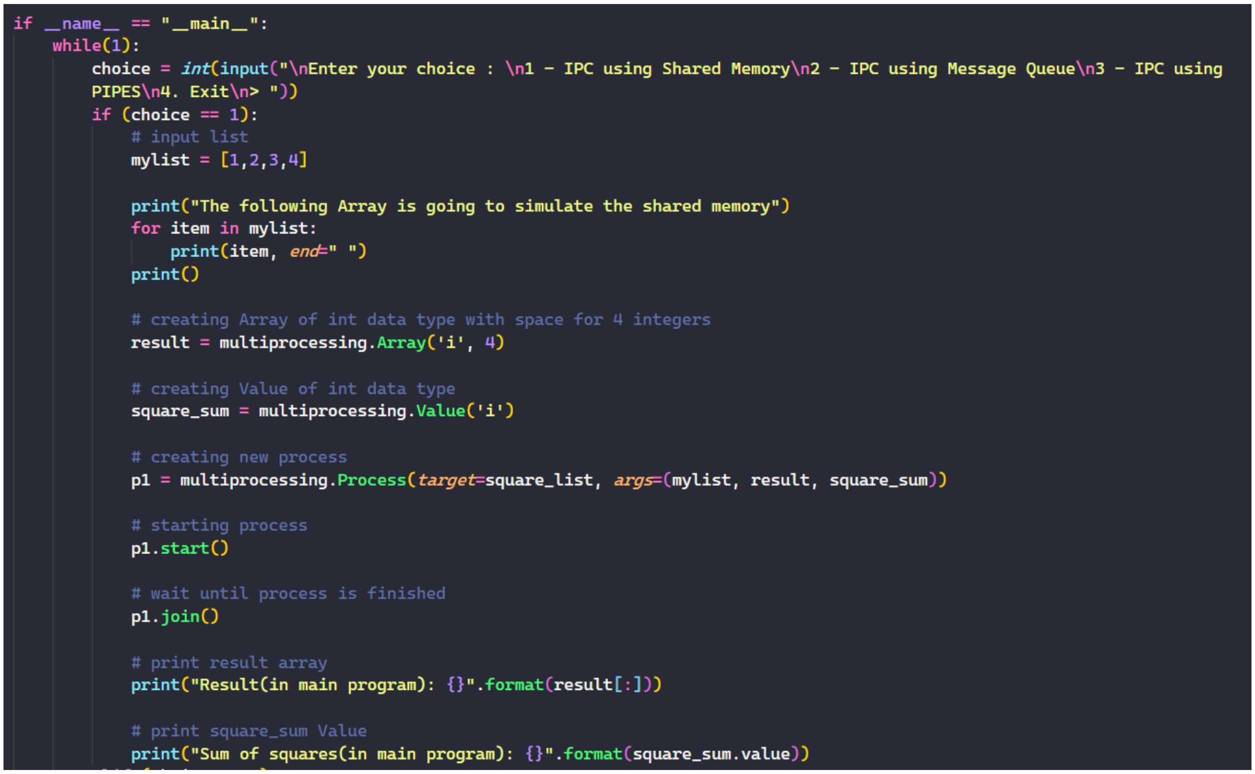
ALGORITHM

1. Start
2. Choose
    a. 1 : IPC Using shared memory
    b. 2 : IPC using Message queue
    c. 3 : IPC using pipes
3. If 1
    a. Create a list to simulate a shared memory
    b. Create a new process
    c. Let the process access the list
    d. Let the main program access the list
4. If 2
    a. Create a list
    b. Create a Queue q
    c. Create two process p1 and p2 that access the queue and perform different functions
    d. Information about the list is accessed using the queue.
5. If 3
    a. Create two process p1 and p2 being the sender and receiver respectively
    b. Create two pipes
    c. Let p1 be the parent pipe and p2 be the child pipe
    d. Data sent through p1 is received at p2.
    e. If data sent is ‘End’, then pipe destroyed.
6. Stop

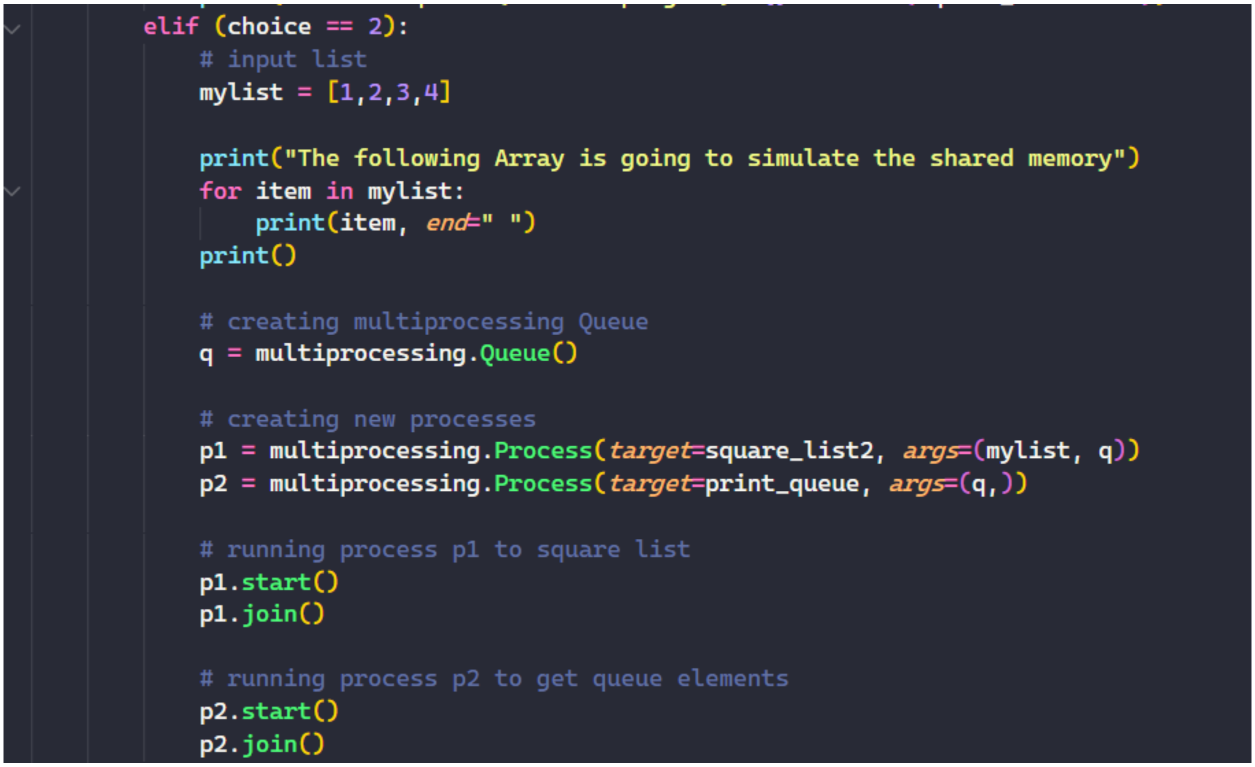
PROGRAM



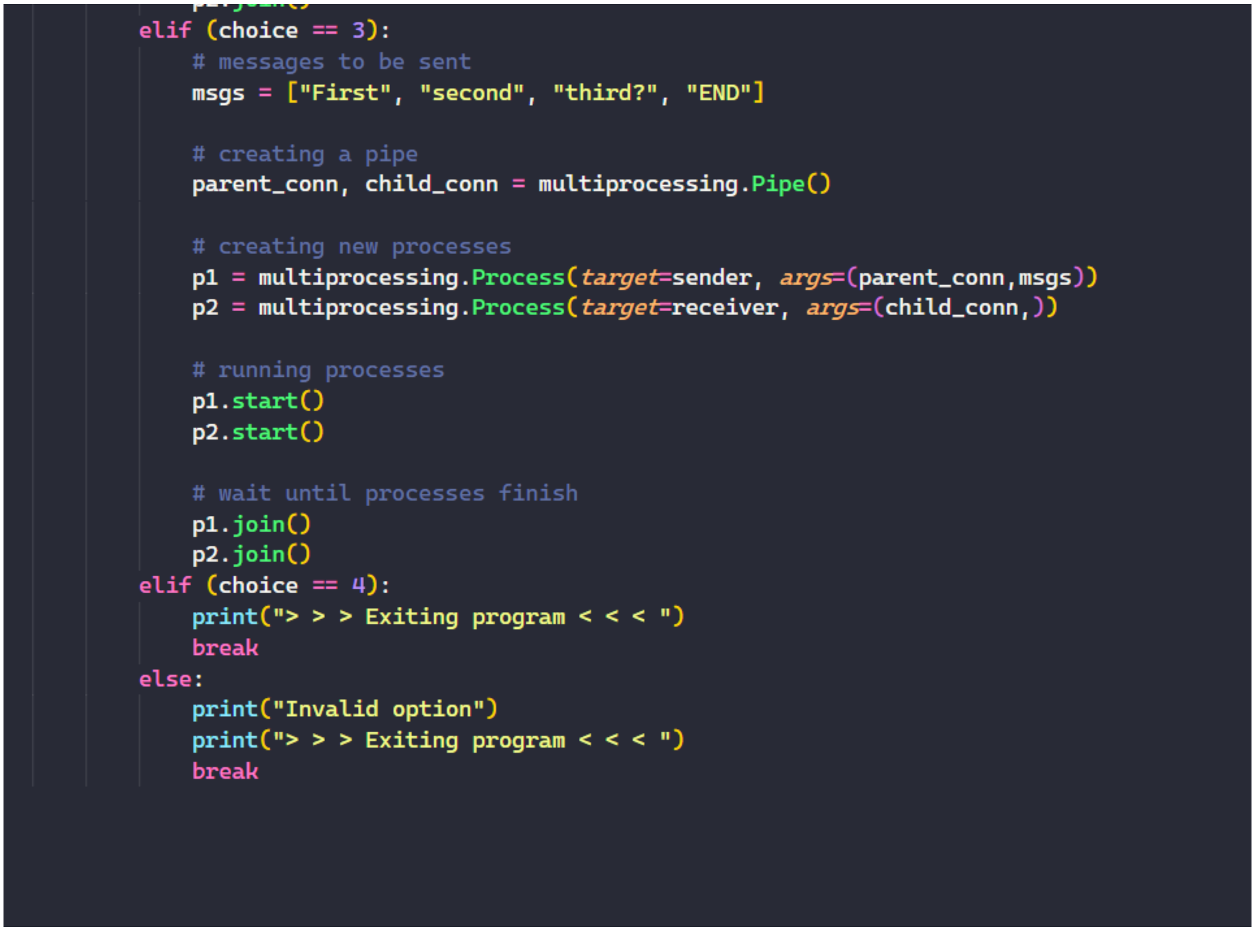
```python
import multiprocessing

def square_list(mylist, result, square_sum):

    # append squares of mylist to result array
    for idx, num in enumerate(mylist):
        result[idx] = num * num

    square_sum.value = sum(result)

    print("Result(in process p1): {}".format(result[:]))

    print("Sum of squares(in process p1): {}".format(square_sum.value))

def square_list2(mylist, q):
    for num in mylist:
        q.put(num * num)

def print_queue(q):
    print("Queue elements:")
    while not q.empty():
        print(q.get())
    print("Queue is now empty!")

def sender(conn, msgs):
    for msg in msgs:
        conn.send(msg)
        print("Sent the message: {}".format(msg))
    conn.close()

def receiver(conn):
    while 1:
        msg = conn.recv()
        if msg == "END":
            break
        print("Received the message: {}".format(msg))
```



```
if __name__ == "__main__":
    while(1):
        choice = int(input("\nEnter your choice : \n1 - IPC using Shared Memory\n2 - IPC using Message Queue\n3 - IPC using
        PIPES\n4. Exit\n> "))
        if (choice == 1):
            # input list
            mylist = [1,2,3,4]

            print("The following Array is going to simulate the shared memory")
            for item in mylist:
                print(item, end=" ")
            print()

            # creating Array of int data type with space for 4 integers
            result = multiprocessing.Array('i', 4)

            # creating Value of int data type
            square_sum = multiprocessing.Value('i')

            # creating new process
            p1 = multiprocessing.Process(target=square_list, args=(mylist, result, square_sum))

            # starting process
            p1.start()

            # wait until process is finished
            p1.join()

            # print result array
            print("Result(in main program): {}".format(result[:]))

            # print square_sum Value
            print("Sum of squares(in main program): {}".format(square_sum.value))
```

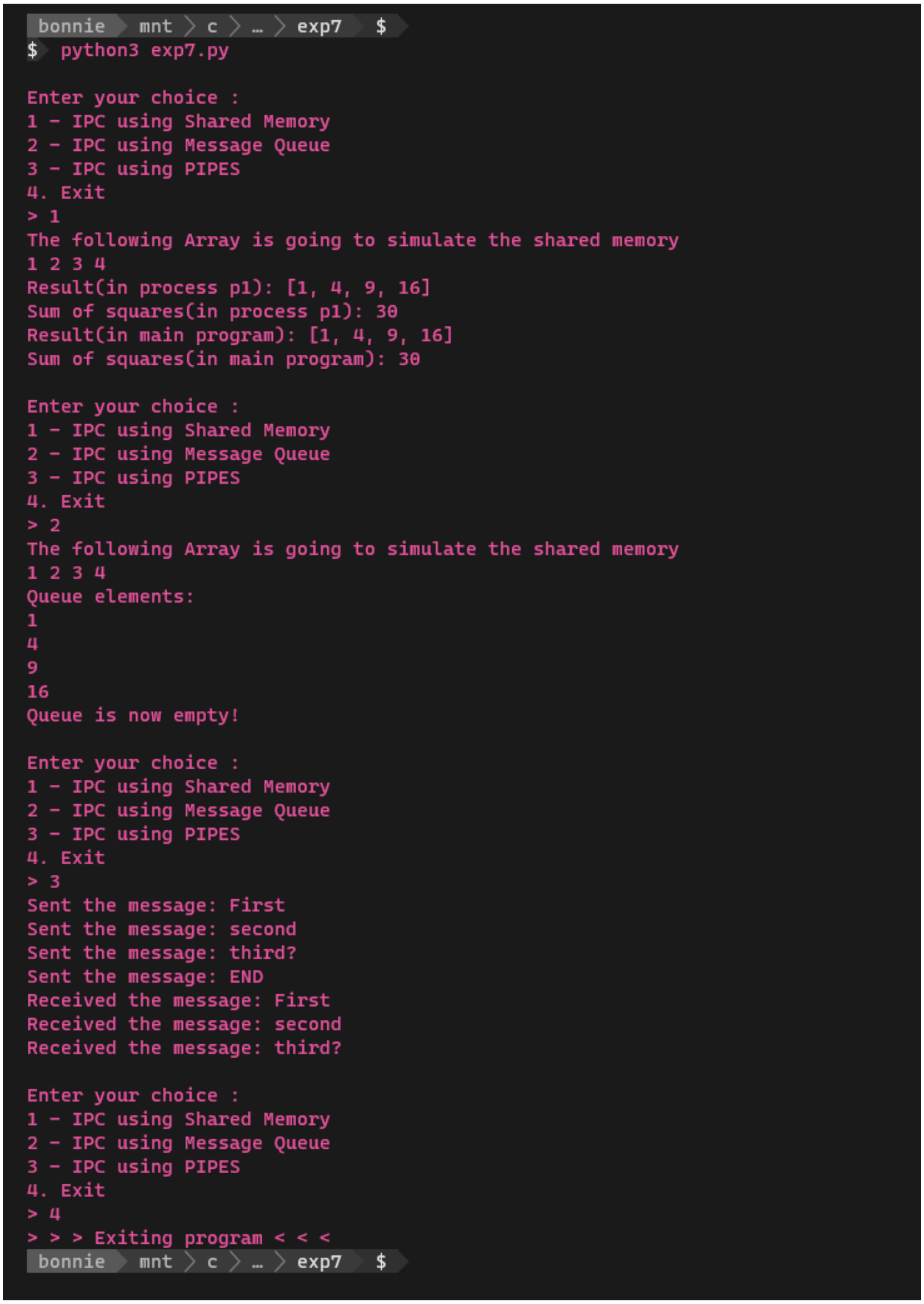


```
        elif (choice == 2):
            # input list
            mylist = [1,2,3,4]

            print("The following Array is going to simulate the shared memory")
            for item in mylist:
                print(item, end=" ")
            print()

            # creating multiprocessing Queue
            q = multiprocessing.Queue()

            # creating new processes
            p1 = multiprocessing.Process(target=square_list2, args=(mylist, q))
            p2 = multiprocessing.Process(target=print_queue, args=(q,))

            # running process p1 to square list
            p1.start()
            p1.join()

            # running process p2 to get queue elements
            p2.start()
            p2.join()
```



```
            p2.join()
        elif (choice == 3):
            # messages to be sent
            msgs = ["First", "second", "third?", "END"]

            # creating a pipe
            parent_conn, child_conn = multiprocessing.Pipe()

            # creating new processes
            p1 = multiprocessing.Process(target=sender, args=(parent_conn,msgs))
            p2 = multiprocessing.Process(target=receiver, args=(child_conn,))

            # running processes
            p1.start()
            p2.start()

            # wait until processes finish
            p1.join()
            p2.join()
        elif (choice == 4):
            print("> > > Exiting program < < < ")
            break
        else:
            print("Invalid option")
            print("> > > Exiting program < < < ")
            break
```

OUTPUT



```
bonnie  mnt  c  …  exp7  $
$ python3 exp7.py

Enter your choice :
1 - IPC using Shared Memory
2 - IPC using Message Queue
3 - IPC using PIPES
4. Exit
> 1
The following Array is going to simulate the shared memory
1 2 3 4
Result(in process p1): [1, 4, 9, 16]
Sum of squares(in process p1): 30
Result(in main program): [1, 4, 9, 16]
Sum of squares(in main program): 30

Enter your choice :
1 - IPC using Shared Memory
2 - IPC using Message Queue
3 - IPC using PIPES
4. Exit
> 2
The following Array is going to simulate the shared memory
1 2 3 4
Queue elements:
1
4
9
16
Queue is now empty!

Enter your choice :
1 - IPC using Shared Memory
2 - IPC using Message Queue
3 - IPC using PIPES
4. Exit
> 3
Sent the message: First
Sent the message: second
Sent the message: third?
Sent the message: END
Received the message: First
Received the message: second
Received the message: third?

Enter your choice :
1 - IPC using Shared Memory
2 - IPC using Message Queue
3 - IPC using PIPES
4. Exit
> 4
> > > Exiting program < < <
bonnie  mnt  c  …  exp7  $
```

RESULT

The python program to implement IPC using message queues, shared memory and pipes have been executed and verified successfully.