Experiment No. 7

Date - 02/06/2021

Inter-process Communication

Aim:

To write a program to Implement Inter-process communication using PIPE, MESSAGE QUEUE and SHARED MEMORY.

THEORY:

Inter-process communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that some event has occurred or the transferring of data from one process to another.

Approaches:

1. Pipe: A pipe is a data channel that is unidirectional. Two pipes can be used to create a two way data channel between two processes. This uses standard input and output methods. Pipes are used in all POSIX systems as well as Windows operating systems.The two different types of pipes are ordinary pipes and named pipes. Ordinary pipes only allow one way communication. For two way communication, two pipes are required. Ordinary pipes have a parent child relationship between the processes as the pipes can only be accessed by processes that created or inherited them.

2. Message Queue: Multiple processes can read and write data to the message queue without being connected to each other. Messages are stored in the queue until their recipient retrieves them. Message queues are quite useful for interprocess communication and are used by most operating systems.

Provides two operations:

I. Send (message)- message size fixed or variable
II. Received (message)

3. Shared Memory: Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes. This type of memory is protected from each other by synchronizing access across all the processes. Shared memory is the memory that can be simultaneously accessed by multiple processes. This is done so that the processes can communicate with each other. All POSIX systems, as well as Windows operating systems use shared memory.

Algorithm

a) Pipe

1. Start
2. Import multiprocessing module
3. Input the data file to be sent and end of file is marked by 0
4. Create process p1 and p2
5. Create pipe
6. Send the data through the pipe at one end and receive through the other end
7. Display the send and received messages
8. Stop

b) Message Queue

1. Start
2. Import multiprocessing module
3. Input the data file
4. Create process p1 and p2
5. Define the functions for inputting to queue and printing
6. Define queue q
7. Assign the processes to the corresponding functions
8. Print the queue
9. Stop

c) Shared Memory

1. Start
2. Import multiprocessing module
3. Input the data file
4. Create process p1
5. Assign the main process as second process
6. Create the array named result and variable sum
7. Define function mult_ten and assign it to p1
8. Display the array and value as p1 and main program
9. Stop

Program

a) Pipe

```
import multiprocessing


def sender(conn, msgs):
        for msg in msgs:
                conn.send(msg)
                print("Messages Sent: {}".format(msg))
        conn.close()
 def receiver(conn):
        while True:
                msg = conn.recv()
                if msg == 0:
                        break
                print("Messages Received: {}".format(msg))
msgs = []
n = int(input("Enter dataset size: "))
for i in range(n):
        a = int(input("Enter the data: "))
        msgs.append(a)
parent_conn, child_conn = multiprocessing.Pipe()
p1 = multiprocessing.Process(target=sender, args=(parent_conn,msgs))
p2 = multiprocessing.Process(target=receiver, args=(child_conn,))
p1.start()
p2.start()
p1.join()
p2.join()
```

b) Message Queue

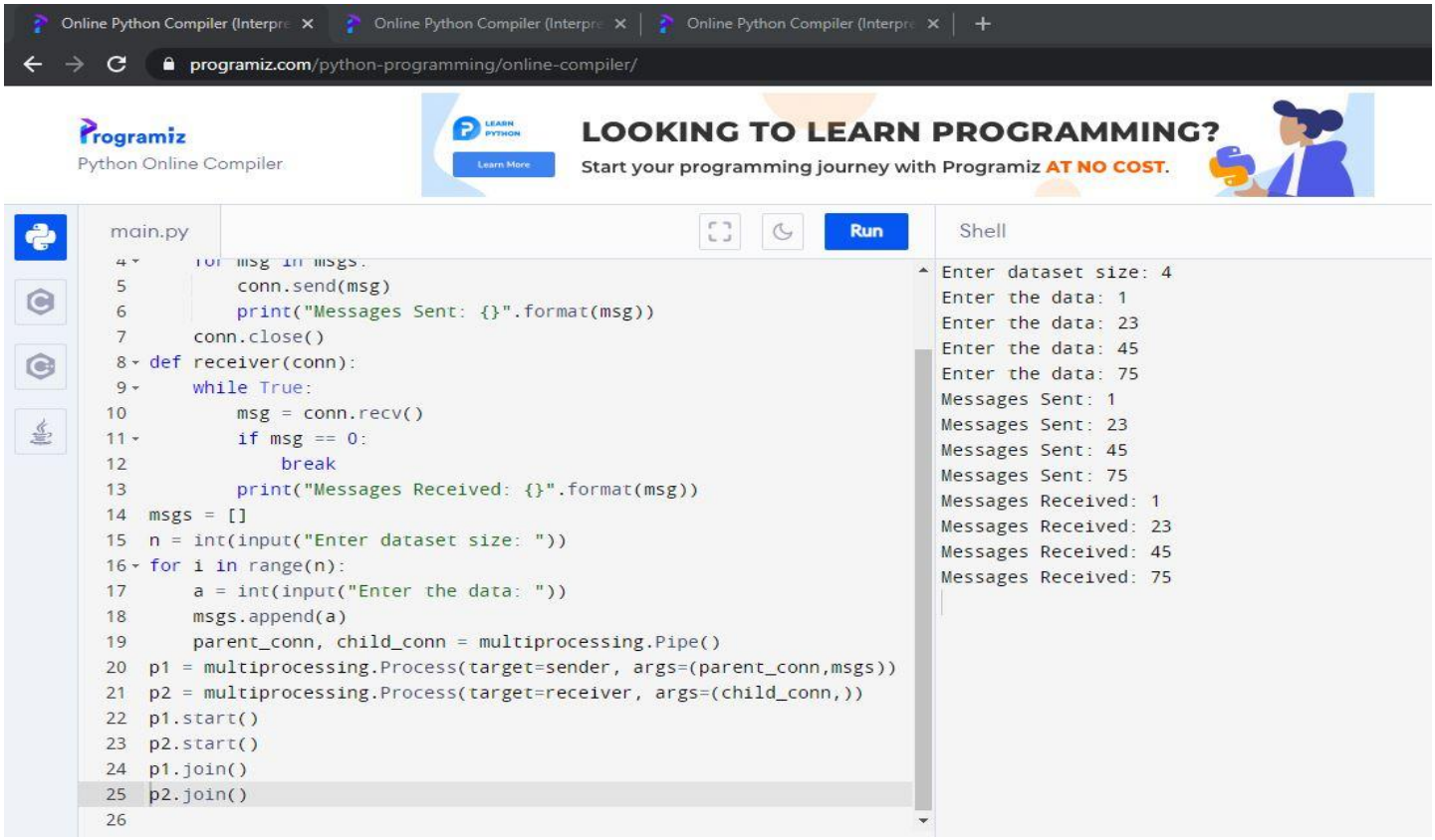
```
import multiprocessing


def mult_ten(list, q):
        print("Queue = list * 10")

        for num in list:

                q.put(num * 10)

def print_queue(q):

        print("Queue elements:")

        while not q.empty():

                print(q.get())

if __name__ == "__main__":

        list= []

n = int(input("Enter the datasize: "))

for i in range(n):

        a = int(input("Enter the data: "))

        list.append(a)

q = multiprocessing.Queue()

p1 = multiprocessing.Process(target=mult_ten, args=(list, q))
p2 = multiprocessing.Process(target=print_queue, args=(q,))
p1.start()

p1.join()

p2.start()

p2.join()
```

c) Shared Memory

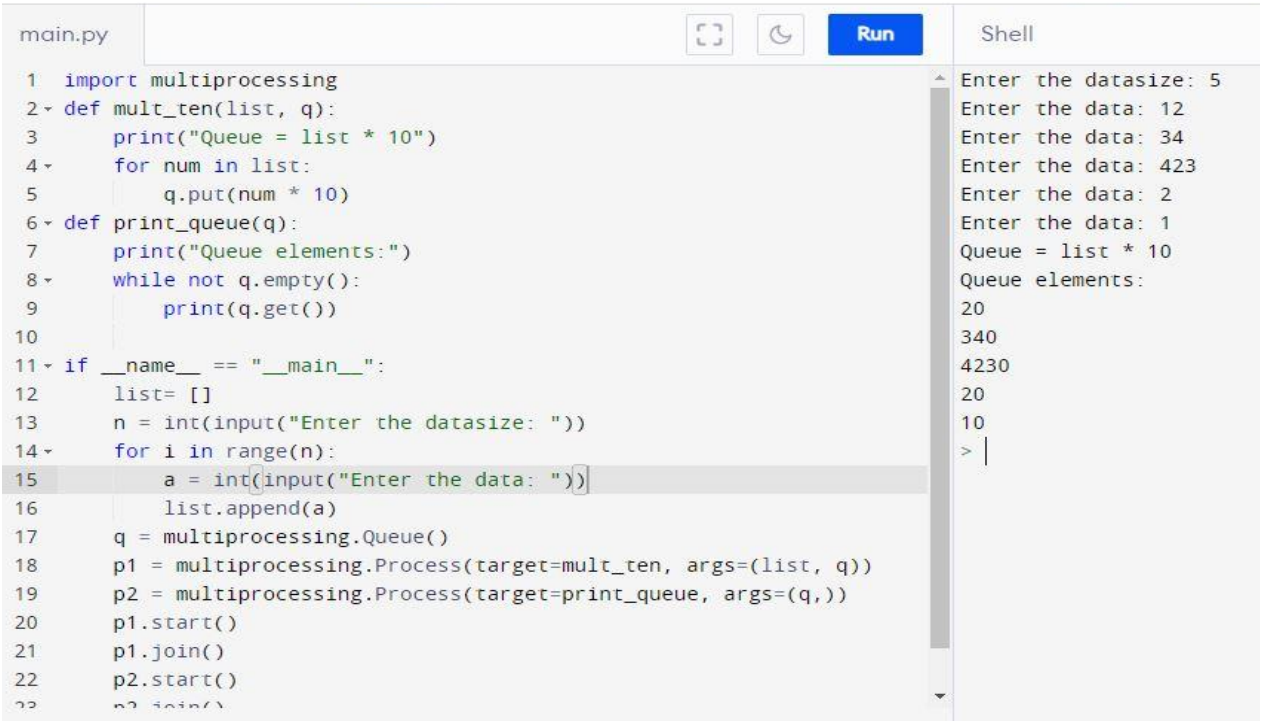
```
import multiprocessing

def mult_ten(list, result, ssum):
	for idx, num in enumerate(list):

		result[idx] = num * 10

		ssum.value = sum(result)

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

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

if __name__ == "__main__":
	list = []

	n = int(input("Enter the datasize: "))

	for i in range(n):

		a = int(input("Enter the data: "))

		list.append(a)

		result = multiprocessing.Array('i', n)

		ssum = multiprocessing.Value('i')

		p1 = multiprocessing.Process(target=mult_ten, args=(list, result, ssum))

		p1.start()
		p1.join()

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

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

Output

a) Pipe



b) Message Queue



c) Shared Memory

main.py

Run

Shell

```
import multiprocessing
def mult_ten(list, result, ssum):
    for idx, num in enumerate(list):
        result[idx] = num * 10
        ssum.value = sum(result)
    print("Result(in process p1): {}".format(result[:]))
    print("Sum of squares(in process p1): {}".format(ssum.value))

if __name__ == "__main__":
    list = []
    n = int(input("Enter the datasize: "))
    for i in range(n):
        a = int(input("Enter the data: "))
        list.append(a)
    result = multiprocessing.Array('i', n)
    ssum = multiprocessing.Value('i')
    p1 = multiprocessing.Process(target=mult_ten, args=(list, result,
        ssum))
    p1.start()
    p1.join()
    print("Result(in main program): {}".format(result[:]))
    print("Sum of squares(in main program): {}".format(ssum.value))
```

```
Enter the datasize: 4
Enter the data: 22
Enter the data: 33
4Enter the data: 4
Enter the data: 55
Result(in process p1): [220, 330, 40, 550]
Sum of squares(in process p1): 1140
Result(in main program): [220, 330, 40, 550]
Sum of squares(in main program): 1140
>
```

Result

Successfully executed a program to implement Inter-process communication using PIPE, MESSAGE QUEUE and SHARED MEMORY.