# Practical No 2

## AIM : Process Communication using Message Passing.

### 1. Use message queues/pipes to solve the producer-consumer problem

```
import multiprocessing
import time
import random

# Producer Function
def producer(queue, n_items):
    for i in range(n_items):
        item = random.randint(1, 100)
        queue.put(item)  # Blocking put
        print(f"Produced: {item}")
        time.sleep(random.uniform(0.1, 0.5))

# Consumer Function
def consumer(queue, n_items):
    for i in range(n_items):
        item = queue.get()  # Blocking get
        print(f"Consumed: {item}")
        time.sleep(random.uniform(0.1, 0.5))

if __name__ == "__main__":
    N_ITEMS = 10
    queue = multiprocessing.Queue(maxsize=5)  # Limited size for demo

    p = multiprocessing.Process(target=producer, args=(queue, N_ITEMS))
    c = multiprocessing.Process(target=consumer, args=(queue, N_ITEMS))

    p.start()
    c.start()
    p.join()
    c.join()
```

```
Produced: 15
Consumed: 15
Produced: 99
Consumed: 99
Produced: 45
Consumed: 45
Produced: 77
Consumed: 77
Produced: 81
Consumed: 81
Produced: 71
Consumed: 71
Produced: 92
Consumed: 92
```

## 2. Blocking Version of Producer-Consumer Using multiprocessing-Queue

```
import multiprocessing
import time
import random
import queue  # For Full/Empty exceptions

# Producer Function (Blocking with try/except)
def producer(queue_obj, n_items):
    for i in range(n_items):
        item = random.randint(1, 100)
        try:
            queue_obj.put(item)  # Blocking put
            print(f"Produced: {item}")
        except queue.Full:
            print("Queue is full. Skipping item.")
        time.sleep(random.uniform(0.1, 0.5))

# Consumer Function (Blocking with try/except)
def consumer(queue_obj, n_items):
    consumed = 0
    while consumed < n_items:
        try:
            item = queue_obj.get()  # Blocking get
            print(f"Consumed: {item}")
            consumed += 1
        except queue.Empty:
            print("Queue is empty. Waiting...")
        time.sleep(random.uniform(0.1, 0.5))

if __name__ == "__main__":
    N_ITEMS = 10
    q = multiprocessing.Queue(maxsize=5)

    p = multiprocessing.Process(target=producer, args=(q, N_ITEMS))
    c = multiprocessing.Process(target=consumer, args=(q, N_ITEMS))

    p.start()
    c.start()
    p.join()
    c.join()
```

```
Produced: 19
Consumed: 19
Produced: 7
Consumed: 7
Produced: 92
Consumed: 92
Produced: 45
Consumed: 45
Produced: 25
Consumed: 25
Produced: 86
Consumed: 86
Produced: 99
Consumed: 99
```

### 3. Non Blocking Version of Producer-Consumer Using multiprocessing-Queue

```
import multiprocessing
import time
import random
import queue  # For Full/Empty exceptions

# Producer Function (Non-blocking put)
def producer(queue_obj, n_items):
    for i in range(n_items):
        item = random.randint(1, 100)
        try:
            queue_obj.put(item, block=False)  # Non-blocking put
            print(f"Produced: {item}")
        except queue.Full:
            print("Queue is full. Skipping item.")
        time.sleep(random.uniform(0.1, 0.5))

# Consumer Function (Non-blocking get)
def consumer(queue_obj, n_items):
    consumed = 0
    while consumed < n_items:
        try:
            item = queue_obj.get(block=False)  # Non-blocking get
            print(f"Consumed: {item}")
            consumed += 1
        except queue.Empty:
            print("Queue is empty. Waiting...")
        time.sleep(random.uniform(0.1, 0.5))

if __name__ == "__main__":
    N_ITEMS = 10
    q = multiprocessing.Queue(maxsize=5)

    p = multiprocessing.Process(target=producer, args=(q, N_ITEMS))
    c = multiprocessing.Process(target=consumer, args=(q, N_ITEMS))

    p.start()
    c.start()
    p.join()
    c.join()
```

```
Produced: 44
Consumed: 44
Produced: 14
Produced: 19
Consumed: 14
Produced: 69
Consumed: 19
Consumed: 69
Produced: 49
Consumed: 49
Queue is empty. Waiting...
Produced: 21
Consumed: 21
Produced: 16
Consumed: 16
```