|  |  |  |  |
| --- | --- | --- | --- |
| **Fr. Conceicao Rodrigues College of Engineering**  **Department of Computer Engineering** | | | |
| **Student’s Roll No** |  | **Students Name** |  |
| **Date of Performance** |  | **SE Computer – Div** | **A / B** |

**Aim:** Study Multiprocessing and Process Synchronization

**Lab Outcome:**

**CSL403.3:** Understand and apply the concepts of synchronization and deadlocks

**Pre-requirement: Python Programming.**

**Problem Statements:**

1. **WAP to demonstrate how to use lock mechanism to achieve process synchronization.**
2. **WAP to demonstrate the use of Queue mechanism to achieve process synchronization in Producer – Consumer Problem.**

**The outputs should reflect behaviour of processes with and without process synchronization in both techniques.**

**References:**

[**https://www.youtube.com/watch?v=RR4SoktDQAw&list=PL5tcWHG-UPH3SX16DI6EP1FlEibgxkg\_6&index=1**](https://www.youtube.com/watch?v=RR4SoktDQAw&list=PL5tcWHG-UPH3SX16DI6EP1FlEibgxkg_6&index=1)

[**https://www.youtube.com/watch?v=iYJNmuD4McE&list=PL5tcWHG-UPH3SX16DI6EP1FlEibgxkg\_6&index=3**](https://www.youtube.com/watch?v=iYJNmuD4McE&list=PL5tcWHG-UPH3SX16DI6EP1FlEibgxkg_6&index=3)

[**https://www.youtube.com/watch?v=TQx3IfCVvQ0&list=PL5tcWHG-UPH3SX16DI6EP1FlEibgxkg\_6&index=6**](https://www.youtube.com/watch?v=TQx3IfCVvQ0&list=PL5tcWHG-UPH3SX16DI6EP1FlEibgxkg_6&index=6)

|  |  |  |  |
| --- | --- | --- | --- |
| **On time Submission(2)** | **Knowledge of Topic(4)** | **Implementation and Demonstraion(4)** | **Total (10)** |
|  |  |  |  |
| **Signature of Faculty** |  | **Date of Submission** |  |

**Process synchronization with Queue:**

from multiprocessing import Process, Queue

import time

def square(numbers, queue):

    for i in numbers:

        time.sleep(0.01)

        queue.put(i\*i)

def cube (numbers,queue):

    for i in numbers:

        time.sleep(0.01)

        queue.put(i\*i\*i)

if \_\_name\_\_=='\_\_main\_\_':

    numbers=range(5)

    queue=Queue()

    square\_process=Process(target=square,args=(numbers, queue))

    cube\_process=Process(target=cube,args=(numbers, queue))

    square\_process.start()

    cube\_process.start()

    square\_process.join()

    cube\_process.join()

    while not queue.empty():

        print(queue.get())

****

**Process synchronization with Lock:**

import time

from multiprocessing import Process, Lock, Value

def add\_500\_lock(total,lock):

    for i in range(100):

        time.sleep(0.01)

        lock.acquire()

        total.value+=5

        lock.release()

def sub\_500\_lock(total,lock):

    for i in range(100):

        time.sleep(0.01)

        lock.acquire()

        total.value-=5

        lock.release()

if \_\_name\_\_=='\_\_main\_\_':

    total=Value('i',500)

    lock=Lock()

    add\_process=Process(target=add\_500\_lock,args=(total,lock))

    sub\_process=Process(target=sub\_500\_lock,args=(total,lock))

    add\_process.start()

    sub\_process.start()

    add\_process.join()

    sub\_process.join()

    print(total.value)

**OutPut:**

****

**Consumer Producer Problem:**

import threading

import time

# Shared Memory variables

CAPACITY = 10

buffer = [-1 for i in range(CAPACITY)]

in\_index = 0

out\_index = 0

# Declaring Semaphores

mutex = threading.Semaphore()

empty = threading.Semaphore(CAPACITY)

full = threading.Semaphore(0)

# Producer Thread Class

class Producer(threading.Thread):

  def run(self):

    global CAPACITY, buffer, in\_index, out\_index

    global mutex, empty, full

    items\_produced = 0

    counter = 0

    while items\_produced < 20:

      empty.acquire()

      mutex.acquire()

      counter += 1

      buffer[in\_index] = counter

      in\_index = (in\_index + 1)%CAPACITY

      print("Producer produced : ", counter)

      mutex.release()

      full.release()

      time.sleep(1)

      items\_produced += 1

# Consumer Thread Class

class Consumer(threading.Thread):

  def run(self):

    global CAPACITY, buffer, in\_index, out\_index, counter

    global mutex, empty, full

    items\_consumed = 0

    while items\_consumed < 20:

      full.acquire()

      mutex.acquire()

      item = buffer[out\_index]

      out\_index = (out\_index + 1)%CAPACITY

      print("Consumer consumed item : ", item)

      mutex.release()

      empty.release()

      time.sleep(2.5)

      items\_consumed += 1

# Creating Threads

producer = Producer()

consumer = Consumer()

# Starting Threads

consumer.start()

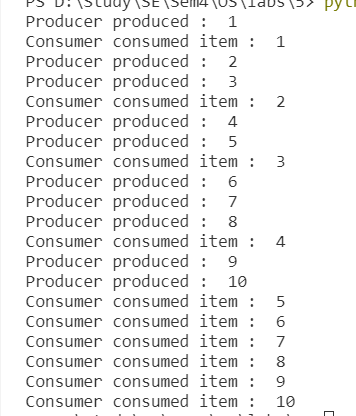
producer.start()

# Waiting for threads to complete

producer.join()

consumer.join()

**Output:**

****