Date - 18/9/2020

Experiment No. 6

Producer Consumer Problem

Aim: To write a program to implement producer-consumer problem.

Theory: In computing, the producer–consumer problem[1][2] (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem, proposed by Dijkstra. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer. The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened. The problem can also be generalized to have multiple producers and consumers.

Algorithm:

Step 1: Start. (Declare the global variables buff and mlock).
Step 2: Input size of the buffer in an integer named N.
Step 3: Input the choice from Producer, Consumer or Exit as integer value to choice.
Step 4: Repeat till program exits (on users input)
    4.1 If choice is 1
        If mlock is 1 and current buffer length is not equal to N
            Call the producer function with argument 1
        Else Print "Buffer is full"
    4.2 Else if choice is 2
        If mlock is 1 and current buffer length is not = 0
            Call the consumer function with argument 1
        Else Print "Buffer is empty"
    4.3 Else Exit the program
Step 5: Stop the program.

Subroutines

1. *wait(s)*
    Step 1: Start.
    Step 2: Try
        Repeat till s<0
            pass
        s = s-1
        Return the value in s
    If error occurs Print "Integer input only."
    Step 3: Stop.
2. *signal(s)*
    Step 1: Start.
    Step 2: Try
        s = s+1
        Return the value in s
    If error occurs Print "Integer input only."
    Step 3: Stop.

3. *producer(n)*
    Step 1: Start. (Use the global variables buff and mlock)
    Step 2: mlock = call wait function with value in mlock and store the return value.
    Step 3: buff = buff + [n] (affects buff's size)
    Step 4: mlock = call signal function with value in mlock and store the return value.
    Step 5: Print "Process added to buffer"
    Step 6: Stop.

4. *consumer(n)*
    Step 1: Start. (Use the global variables buff and mlock)
    Step 2: mlock = call wait function with value in mlock and store the return value.
    Step 3: value = remove and store the value in buff at index *n* (affects buff's size)
    Step 4: mlock = call signal function with value in mlock and store the return value.
    Step 5: Print "Process removed from buffer."
    Step 6: Stop.

Program:

```
import os
def screen_clear():
    # for mac and linux(here, os.name is 'posix')
    if os.name == 'posix':
        _ = os.system('clear')
    else:
        # for windows platform
        _ = os.system('cls')

screen_clear()
N = int(input("Enter the size of Buffer : "))

def wait(s):
    try:
        while(s<0):
            pass
        s-=1
        return(s)
    except:
        print("Integer input only.")

def signal(s):
    try:
        s+=1
        return(s)
    except:
        print("Integer input only.")

def producer(n):
    global buff, mlock
    mlock = wait(mlock)
    buff += [n]
    mlock = signal(mlock)
    print("Process added to buffer")

def consumer(n):
    global buff,mlock
    mlock = wait(mlock)
    value = buff.pop(buff.index(n))
    mlock = signal(mlock)
    print("Process removed from buffer")

```

```
def main():
    choice = int(input("\n1.Produce\n2.Consume\n3.Exit:\n"))
    if(choice==1):
        if(mlock==1 and len(buff)!=N):
            producer(1)
        else:
            print("Buffer is full\n")
    elif(choice==2):
        if(mlock==1 and len(buff)!=0):
            consumer(1)
        else:
            print("Buffer is empty\n")
    else:
        exit()

if __name__ == '__main__':
    mlock = 1
    buff  = []
    while(1):
        main()
```

```
import os
def screen_clear():
  # for mac and linux(here, os.name is 'posix')
  if os.name == 'posix':
    _ = os.system('clear')
  else:
    # for windows platform
    _ = os.system('cls')

screen_clear()
N = int(input("Enter the size of Buffer : "))

def wait(s):
        try:
                while(s<0):
                        pass
                s-=1
                return(s)
        except:
                print("Integer input only.")

def signal(s):
        try:
                s+=1
                return(s)
        except:
                print("Integer input only.")
```

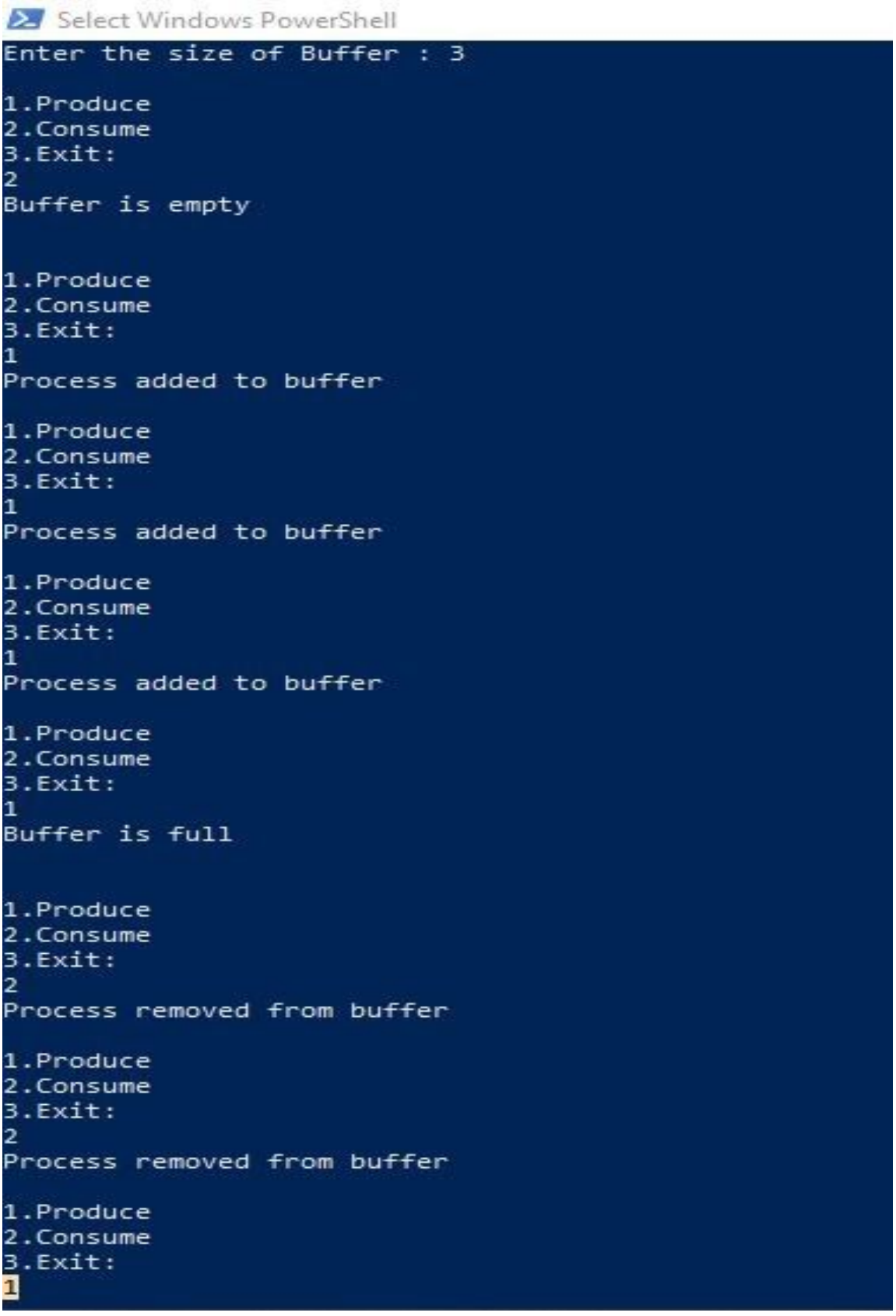
```
def producer(n):
        global buff, mlock
        mlock = wait(mlock)
        buff += [n]
        mlock = signal(mlock)
        print("Process added to buffer")

def consumer(n):
        global buff,mlock
        mlock = wait(mlock)
        value = buff.pop(buff.index(n))
        mlock = signal(mlock)
        print("Process removed from buffer")

def main():
        choice = int(input("\n1.Produce\n2.Consume\n3.Exit:\n"))
        if(choice==1):
                if(mlock==1 and len(buff)!=N):
                        producer(1)
                else:
                        print("Buffer is full\n")
        elif(choice==2):
                if(mlock==1 and len(buff)!=0):
                        consumer(1)
                else:
                        print("Buffer is empty\n")
        else:
                exit()

if __name__ == '__main__':
   mlock = 1
   buff  = []
   while(1):
      main()
```

OR

```
from threading import Thread, Condition
import time
import random

queue = []
MAX_NUM = 10
condition = Condition()
```

```
class ProducerThread(Thread):
    def run(self):
        nums = range(5)
        global queue
        while True:
            condition.acquire()
            if len(queue) == MAX_NUM:
                print "Queue full, producer is waiting"
                condition.wait()
                print "Space in queue, Consumer notified the producer"
            num = random.choice(nums)
            queue.append(num)
            print "Produced", num
            condition.notify()
            condition.release()
            time.sleep(random.random())


class ConsumerThread(Thread):
    def run(self):
        global queue
        while True:
            condition.acquire()
            if not queue:
                print "Nothing in queue, consumer is waiting"
                condition.wait()
                print "Producer added something to queue and notified the consumer"
            num = queue.pop(0)
            print "Consumed", num
            condition.notify()
            condition.release()
            time.sleep(random.random())


ProducerThread().start()
ConsumerThread().start()
```

Output:


Select Windows PowerShell
Enter the size of Buffer : 3
1.Produce
2.Consume
3.Exit:
2
Buffer is empty
1.Produce
2.Consume
3.Exit:
1
Process added to buffer
1.Produce
2.Consume
3.Exit:
1
Process added to buffer
1.Produce
2.Consume
3.Exit:
1
Process added to buffer
1.Produce
2.Consume
3.Exit:
1
Buffer is full
1.Produce
2.Consume
3.Exit:
2
Process removed from buffer
1.Produce
2.Consume
3.Exit:
2
Process removed from buffer
1.Produce
2.Consume
3.Exit:
1

```
Process added to buffer

1.Produce
2.Consume
3.Exit:
2
Process removed from buffer

1.Produce
2.Consume
3.Exit:
2
Process removed from buffer

1.Produce
2.Consume
3.Exit:
2
Buffer is empty


1.Produce
2.Consume
3.Exit:
3
PS G:\S5 CS\SS Lab> ^C
```

Enter the size of Buffer : 3

1.Produce
2.Consume
3.Exit:
2
Buffer is empty

1.Produce
2.Consume
3.Exit:
1
Process added to buffer

1.Produce
2.Consume
3.Exit:
1
Process added to buffer

1.Produce
2.Consume
3.Exit:
1
Process added to buffer

1.Produce
2.Consume
3.Exit:
1
Buffer is full

1.Produce
2.Consume
3.Exit:
2
Process removed from buffer

1.Produce
2.Consume
3.Exit:
2
Process removed from buffer

1.Produce
2.Consume
3.Exit:
1
Process added to buffer

1.Produce
2.Consume
3.Exit:
2
Process removed from buffer

1.Produce
2.Consume
3.Exit:
2
Process removed from buffer

1.Produce
2.Consume
3.Exit:
2
Buffer is empty

1.Produce
2.Consume
3.Exit:
3
PS G:\S5 CS\SS Lab>

OR

Produced 0
Consumed 0
Produced 0
Produced 4
Consumed 0
Consumed 4
Nothing in queue, consumer is waiting
Produced 4
Producer added something to queue and notified the consumer
Consumed 4
Produced 3
Produced 2
Consumed 3

Result:

The program to simulate the producer-consumer problem has been implemented successfully.