Batch: A2 **Roll No.: 1911027**

Experiment No. 05

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE: Implementation of Process synchronization algorithms using semaphore - producer consumer problem , reader-writers problem

---

AIM: Implementation of Process synchronization algorithms using semaphore - producer consumer problem, reader-writers problem

---

Expected Outcome of Experiment:

CO 3. To understand the concepts of process synchronization and deadlock.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate "Operating Systems" McGraw Hill Third Edition.**
3. **William Stallings, "Operating System Internal & Design Principles", Pearson.**
4. **Andrew S. Tanenbaum, "Modern Operating System", Prentice Hall.**

Pre Lab/ Prior Concepts:

Knowledge of Concurrency, Mutual Exclusion, Synchronization, Deadlock, Starvation.

Description of the chosen process synchronization algorithm:

Producer-Consumer problem:

In computing, the producer–consumer problem is a classic example of a multi-process synchronization problem. In the first version of the problem, there are two cyclic processes, a producer and a consumer, which share a common, fixed-size buffer used as a queue. The producer repeatedly generates data and writes it into the buffer. The consumer repeatedly reads the data in the buffer, removing it in the course of reading it, and using that data in some way. In the first version of the problem, with an unbounded buffer, the problem is how to design the producer and consumer code so that, in their exchange of data, no data is lost or duplicated, data is read by the consumer in the order it is written by the producer, and both processes make as much progress as possible. In the later formulation of the problem, author proposed multiple producers and consumers sharing a finite collection of buffers. This added the additional problem of preventing producers from trying to write into buffers when all were full, and trying to prevent consumers from reading a buffer when all were empty. The first case to consider is the one in which there is a single producer and a single consumer, and there is a finite-size 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 producer should produce data only when the buffer is not full. If the buffer is full, then the producer shouldn't be allowed to put any data into the buffer. The consumer should consume data only when the buffer is not empty. If the buffer is empty, then the consumer shouldn't be allowed to take any data from the buffer. The producer and consumer should not access the buffer at the same time.

Solution using semaphore:

The problem which is described earlier can be solved by using 3 semaphores they are described as follows:

Semaphore mutex: This semaphore variable is used to achieve mutual exclusion between processes. By using this variable, either Producer or Consumer will be allowed to use or access the shared buffer at a particular time. This variable is set to 1 initially.

Semaphore empty: This semaphore variable is used to define the empty space in the buffer. Initially, it is set to the whole space of the buffer i.e. "n" because the buffer is initially empty.

Semaphore full: This semaphore variable is used to define the space that is filled by the producer. Initially, it is set to "0" because there is no space filled by the producer initially.

By using the above three semaphore variables and by using the wait() and signal() function, we can solve our problem(the wait() function decreases the semaphore variable by 1 and the signal() function increases the semaphore variable by 1).

The pseudo code for producer can be given by:

```
void producer() {
   while(True) {
      produce() // function to produce data by the producer.
      wait(empty) // will reduce the value of the semaphore variable "empty" by one
      wait(mutex) // is used to set the semaphore variable "mutex" to "0" so that no
                        other process can enter into the critical section.
      append() // function to append the newly produced data in the buffer.
      signal(mutex) // is used to set the semaphore variable "mutex" to "1" so that other processes can come into the critical section
      signal(full)
   }
}
```

The pseudo code for consumer can be given by:

```
void consumer() {
   while(True) {
      wait(full) // decrease semaphore variable full by 1
```

```
        wait(mutex) // set mutex to 0 so that producer cannot access critical section
        take() // take an item from the buffer (consume it)
        signal(mutex) // set mutex to 1 so that producer now can go in critical section
        signal(empty) // increase semaphore variable empty by 1
        use()
    }
}
```

Above two algorithms can be used to overcome producer consumer problem. Mutex will ensure mutual exclusion. Empty will ensure that if buffer is full producer will not be able to insert any new items. Full will ensure that if buffer is empty consumer will not be able to consume anything.

Reader-Writer problem:

The readers-writers problem relates to an object such as a file that is shared between multiple processes. Some of these processes are readers i.e. they only want to read the data from the object and some of the processes are writers i.e. they want to write into the object. The readers-writers problem is used to manage synchronization so that there are no problems with the object data. For example - If two readers access the object at the same time there is no problem. However if two writers or a reader and writer access the object at the same time, there may be problems. Problem parameters:

- One set of data is shared among a number of processes.
- Once a writer is ready, it performs its write. Only one writer may write at a time.
- If a process is writing, no other process can read it.
- If at least one reader is reading, no other process can write.
- Readers may not write and only read.

Solution using semaphore:

To solve the above situation, a writer should get exclusive access to an object i.e. when a writer is accessing the object, no reader or writer may access it. However, multiple readers can access the object at the same time. This can be implemented using semaphores. Three semaphore variables are used: mutex, wrt, readcount to implement solution:

- semaphore mutex, wrt: semaphore mutex is used to ensure mutual exclusion when readcount is updated i.e. when any reader enters or exit from the critical section and semaphore wrt is used by both readers and writers.
- int readcnt: This tells the number of processes performing read in the critical section, initially 0.

Functions for sempahore:

– wait() : decrements the semaphore value.

– signal() : increments the semaphore value.

The pseudo code for writer can be given by:

```
do {

    // writer requests for critical section

    wait(wrt);

    // performs the write

    // leaves the critical section

    signal(wrt);

} while(true);
```

Writer requests the entry to critical section. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting. It exits the critical section.

The pseudo code for reader can be given by:

```
do {

  // Reader wants to enter the critical section

  wait(mutex);

  // The number of readers has now increased by 1

  readcount++;

  // there is atleast one reader in the critical section

  // this ensure no writer can enter if there is even one reader

  // thus we give preference to readers here

  if (readcnt==1)

    wait(wrt);

  // other readers can enter while this current reader is inside

  // the critical section

  signal(mutex);

  // current reader performs reading here

  wait(mutex);   // a reader wants to leave

  readcount--;

  // that is, no reader is left in the critical section,
```

```
  if (readcount == 0)
     signal(wrt);        // writers can enter
  signal(mutex); // reader leave
} while(true);
```

Reader requests the entry to critical section. If allowed: it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the wrt semaphore to restrict the entry of writers if any reader is inside. It then, signals mutex as any other reader is allowed to enter while others are already reading. After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore "wrt" as now, writer can enter the critical section. If not allowed, it keeps on waiting. Thus, the semaphore 'wrt' is queued on both readers and writers in a manner such that preference is given to readers if writers are also there. Thus, no reader is waiting simply because a writer has requested to enter the critical section.

Implementation details: (printout of code)

Producer consumer problem:
Code:

```
import random
import time
import threading
from threading import Thread
def wait(s):
    return s-1

def signal(s):
    return s+1

def producer():
    global mutex
    global empty
    global full
```

```
    global temp
    global pr
    global cn
    n=0
    while(True):
        n=n+1
        if(mutex==0):
            if(cn==False):
                print("Producer: Consumer in critical section
wait......")
                print("------------------------------------------------
-------------------------------------------------")
                if(n==5):
                    break
            elif(pr==False):
                print("Producer: Producer in critical section
wait......")
                print("------------------------------------------------
-------------------------------------------------")
                if(n==5):
                    break
            time.sleep(2)
        elif(empty==0):
            print("Producer: Buffer is full!!!!")
            print("------------------------------------------------
-------------------------------------------")
            if(n==5):
                break
            time.sleep(2)
        else:
            pr=False
            empty=wait(empty)
            mutex=wait(mutex)
            temp=(temp+1)%n
            buffer.append(temp)
            print("Lock acquired by producer!!!")
            print("Producer producing an item...")
            print("------------------------------------------------
-------------------------------------------")
            time.sleep(2)
            pr=True
```

```
            full=signal(full)
            mutex=signal(mutex)
            print("Producer: Item produced: ",temp,"Buffer: ",buffer)
            print("Lock released by producer!!!")
            print("------------------------------------------------------
-----------------------------------------")
            if(n==5):
                break

def consumer():
    global mutex
    global empty
    global full
    global temp
    global pr
    global cn
    n=0
    while(True):
        n=n+1
        if(mutex==0):
            if(pr==False):
                print("Consumer: Producer in critical section
wait......")
                print("------------------------------------------------
-------------------------------------------------")
                if(n==5):
                    break
            elif(cn==False):
                print("Consumer: Consumer in critical section
wait......")
                print("------------------------------------------------
-------------------------------------------------")
                if(n==5):
                    break
            time.sleep(2)
        elif(full==0):
            print("Consumer: Buffer is empty!!!!")
            print("------------------------------------------------------
-----------------------------------------")
            if(n==5):
                break
```

```
            time.sleep(2)
        else:
            cn=False
            mutex=wait(mutex)
            full=wait(full)
            item=buffer.pop(0)
            print("Lock acquired by consumer!!!")
            print("Consumer consuming an item ",item,"...")
            print("-----------------------------------------------------------
-------------------------------------------")
            time.sleep(2)
            cn=True
            empty=signal(empty)
            mutex=signal(mutex)
            print("Consumer: Item consumed: ",item,"Buffer: ",buffer)
            print("Lock released by consumer!!!")
            print("-----------------------------------------------------------
-------------------------------------------")
            if(n==5):
                break

print("-----------------------------------------------------------------------
---------------------------------")
print("----------------------------------------PRODUCER-CONSUMER--------
---------------------------------")
print("-----------------------------------------------------------------------
---------------------------------")
n=int(input("Enter buffer size: "))
print("-----------------------------------------------------------------------
---------------------------------")
empty=n
full=0
mutex=1
buffer=[]
temp=0
producerconsumer=[]
pr=True
cn=True
producerconsumer.append(Thread(target=producer, name="Producer"))
producerconsumer.append(Thread(target=consumer, name="Consumer"))
t=0
```

```
temp=-1
for i in producerconsumer:
    i.start()
    time.sleep(1)
```

Output:

```
------------------------------------------------------------------------------------------------------------
----------------------------------------PRODUCER-CONSUMER---------------------------------------
------------------------------------------------------------------------------------------------------------
Enter buffer size: 2
------------------------------------------------------------------------------------------------------------
Lock acquired by producer!!!
Producer producing an item...
------------------------------------------------------------------------------------------------------------
Consumer: Producer in critical section wait......
------------------------------------------------------------------------------------------------------------
Producer: Item produced:  0 Buffer:  [0]
Lock released by producer!!!
------------------------------------------------------------------------------------------------------------
Lock acquired by producer!!!
Producer producing an item...
------------------------------------------------------------------------------------------------------------
Consumer: Producer in critical section wait......
------------------------------------------------------------------------------------------------------------
Producer: Item produced:  1 Buffer:  [0, 1]
Lock released by producer!!!
------------------------------------------------------------------------------------------------------------
Producer: Buffer is full!!!!
------------------------------------------------------------------------------------------------------------
Lock acquired by consumer!!!
Consumer consuming an item  0 ...
------------------------------------------------------------------------------------------------------------
Producer: Consumer in critical section wait......
------------------------------------------------------------------------------------------------------------
Consumer: Item consumed:  0 Buffer:  [1]
Lock released by consumer!!!
------------------------------------------------------------------------------------------------------------
Lock acquired by consumer!!!
Consumer consuming an item  1 ...
------------------------------------------------------------------------------------------------------------
Producer: Consumer in critical section wait......
------------------------------------------------------------------------------------------------------------
Consumer: Item consumed:  1 Buffer:  []
Lock released by consumer!!!
------------------------------------------------------------------------------------------------------------
Consumer: Buffer is empty!!!!
------------------------------------------------------------------------------------------------------------
```

K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Reader writer problem:

Code:

```
import threading
from threading import Thread
import random
import time
def wait(s):
    return s-1
def signal(s):
    return s+1
def reader():
    global mutex
    global readcount
    global wrt
    while(True):
        flag=0
        if(mutex!=0):
            mutex=wait(mutex)
            readcount=readcount+1
            if(readcount==1):
                if(wrt!=0):
                    wrt=wait(wrt)
                else:
                    print("Some write operation is goin on!!!")
                    print("----------------------------------------------
------------------------------------------------------------------------")
                    flag=1
            if(flag==0):
                mutex=signal(mutex)
                print(threading.current_thread().name,": Reading
started!!", "Number of readers : ",readcount)
                print("----------------------------------------------------
------------------------------------------------------------------")
                time.sleep(5)
                mutex=wait(mutex)
                readcount=readcount-1
                print(threading.current_thread().name,": Reading
stopped!!", "Number of readers : ",readcount)
                print("----------------------------------------------------
------------------------------------------------------------------")
                if(readcount==0):
```

```
                    wrt=signal(wrt)
                mutex=signal(mutex)
                break
            else:
                mutex=signal(mutex)
                readcount=readcount-1
        else:
            print("Some write operation going on :
",threading.current_thread().name," waiting!!!")
            print("----------------------------------------------------------
------------------------------------------------------------")
            time.sleep(5)
def writer():
    global wrt
    global mutex
    while(True):
        if(wrt!=0 and mutex!=0):
            wrt=wait(wrt)
            mutex=wait(mutex)
            print(threading.current_thread().name, ": Writing started!!")
            print("----------------------------------------------------------
------------------------------------------------------------")
            time.sleep(5)
            print(threading.current_thread().name, ": Writing stopped!!")
            print("----------------------------------------------------------
------------------------------------------------------------")
            wrt=signal(wrt)
            mutex=signal(mutex)
            break
        else:
            if(wrt==1 and mutex==0):
                print("Some write operation going on :
",threading.current_thread().name," waiting!!!")
                print("----------------------------------------------------------
------------------------------------------------------------")
            elif(wrt==0 and mutex==1):
                print("Some read operation going on :
",threading.current_thread().name," waiting!!!")
                print("----------------------------------------------------------
------------------------------------------------------------")
            else:
```

K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

```
                print("Some write operation going on :
",threading.current_thread().name," waiting!!!")
                print("-----------------------------------------------------
-------------------------------------------------------------------")
            time.sleep(5)
readerwriter=[]
mutex=1
wrt=1
readcount=0
print("-------------------------------------------------------------------
-----------------------------------------------")
print("-----------------------------------------------------READER-
WRITER-----------------------------------------------")
print("-------------------------------------------------------------------
-----------------------------------------------")
n1=int(input("Enter number of readers : "))
print("-------------------------------------------------------------------
-----------------------------------------------")
for i in range(0,n1):
    temp="Reader"+str(i)
    readerwriter.append(Thread(target=reader, name=temp))
n2=int(input("Enter number of writers : "))
print("-------------------------------------------------------------------
-----------------------------------------------")
for i in range(0,n2):
    temp="Writer"+str(i)
    readerwriter.append(Thread(target=writer, name=temp))
random.shuffle(readerwriter)
t=0
for i in readerwriter:
    t=t+1
    if(t==10):
        break
    i.start()
    time.sleep(1)
```

Output:

```
----------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------READER-WRITER---------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------
Enter number of readers : 2
----------------------------------------------------------------------------------------------------------------------------
Enter number of writers : 4
----------------------------------------------------------------------------------------------------------------------------
Writer0 : Writing started!!
----------------------------------------------------------------------------------------------------------------------------
Some write operation going on :  Reader1  waiting!!!
----------------------------------------------------------------------------------------------------------------------------
Some write operation going on :  Writer2  waiting!!!
----------------------------------------------------------------------------------------------------------------------------
Some write operation going on :  Writer3  waiting!!!
----------------------------------------------------------------------------------------------------------------------------
Some write operation going on :  Writer1  waiting!!!
----------------------------------------------------------------------------------------------------------------------------
Writer0 : Writing stopped!!
----------------------------------------------------------------------------------------------------------------------------
Reader0 : Reading started!! Number of readers :  1
----------------------------------------------------------------------------------------------------------------------------
Reader1 : Reading started!! Number of readers :  2
----------------------------------------------------------------------------------------------------------------------------
Some read operation going on :  Writer2  waiting!!!
----------------------------------------------------------------------------------------------------------------------------
Some read operation going on :  Writer3  waiting!!!
----------------------------------------------------------------------------------------------------------------------------
Some read operation going on :  Writer1  waiting!!!
----------------------------------------------------------------------------------------------------------------------------
Reader0 : Reading stopped!! Number of readers :  1
----------------------------------------------------------------------------------------------------------------------------
Reader1 : Reading stopped!! Number of readers :  0
----------------------------------------------------------------------------------------------------------------------------
Writer2 : Writing started!!
----------------------------------------------------------------------------------------------------------------------------
Some write operation going on :  Writer3  waiting!!!
----------------------------------------------------------------------------------------------------------------------------
Some write operation going on :  Writer1  waiting!!!
----------------------------------------------------------------------------------------------------------------------------
Writer2 : Writing stopped!!
----------------------------------------------------------------------------------------------------------------------------
Writer3 : Writing started!!
----------------------------------------------------------------------------------------------------------------------------
Writer3 : Writing started!!
----------------------------------------------------------------------------------------------------------------------------
Some write operation going on :  Writer1  waiting!!!
----------------------------------------------------------------------------------------------------------------------------
Writer3 : Writing stopped!!
----------------------------------------------------------------------------------------------------------------------------
Writer1 : Writing started!!
----------------------------------------------------------------------------------------------------------------------------
Writer1 : Writing stopped!!
----------------------------------------------------------------------------------------------------------------------------
```

Conclusion: By performing this experiment understood the problems invoked during process synchronization. Also understood the use of semaphores to solve process synchronization problems. Used semaphores to solve two well known process synchronization problems named producer consumer problem and reader writer problem. Implemented solutions to both the problems using semaphores in python programming language.

Post Lab Objective Questions

1) A semaphore is a shared integer variable
   a) That can't drop below zero
   b) That can't be more than
   c) That can't drop below one

Ans: a) That can't drop below zero

2) Mutual exclusion can be provided by the
   a) Mute locks
   b) Binary semaphores
   c) Both a and b
   d) None of these

Ans: c) Both a and b

3) A monitor is a module that encapsulates
   a) Shared data structures
   b) Procedures that operate on shared data structure
   c) Synchronization between concurrent procedure invocation
   d) All of the above

Ans: a) All of the above

4) To enable a process to wait within the monitor

a) A condition variable must be declared as condition

b) Condition Variables must be used as Boolean objects

c) Semaphore must be used

d) All of the above

Ans: a) A condition variable must be declared as condition

Date: 24 / 11 / 2021 **Signature of faculty in-charge**