Date - 15/04/2021

Experiment No. 3

Write a program to implement the First Readers-Writers Problem.

Aim **:** *To Write a program to implement the First Readers-Writers Problem.*

Theory

In computer science, the readers-writer problems are examples of a common computing problem in concurrency. There are at least three variations of the problems, which deal with situations in which many threads (small processes which share data) try to access the same shared resource at one time. Some threads may read and some may write, with the constraint that no thread may access the shared resource for either reading or writing while another thread is in the act of writing to it.

In particular, we want to prevent more than one thread modifying the shared resource simultaneously and allow for two or more readers to access the shared resource at the same time. So, we can use the reader-writer lock data structure to overcome this problem.

There is a shared data which can be accessed by multiple processes. There are two types of processes in the system i.e a reader and a writer. Any number of readers can read from the shared resource simultaneously, but only one writer can write to the shared resource. When a writer is writing data to the resource, no other process can access the resource. A writer cannot write to the resource if there are non zero number of readers accessing the resource at that time.

Algorithm **:**

Note:

1. *Signal of semaphore increase the value by one hence releasing the lock*
2. *Wait of semaphore decrease the value by one hence acquiring the lock*
3. *A binary semaphore is either 1 or 0*

Writer:

```
wait(wrt);
    . . .
    writing is performed  //wait (wrt) ensures no other writer access the data
    . . .
signal(wrt);
```

Reader:

```
wait(mutex);
        readcount := readcount + 1;
        if readcount = 1 then
wait(wrt);  //since reader is reading, prevent writing for writer

signal(mutex);
        . . .
        reading is performed
        . . .
        . . .
wait(mutex);
        readcount := readcount - 1;
        if readcount = 0 then signal(wrt); //since no reader is present allow
 writer to write
signal(mutex);
```

Import Libraries for Reader-Writer

First, we need to import the thread library to our project and create a lock variable that is used to solve the Reader-Writer problem.

Along with that, declare the global variable which needs to be accessed while Reading as well as Writing in Synchronous way.

```
import threading as thread

import random

global x

x = 0

lock = thread.Lock()
```

Define Reader Function

Now, as we imported our required libraries to our program, let's create a new function named Reader() which is used to read a global variable in a synchronous way.

In this, before accessing or reading the global variable, we need to ensure that we acquire a lock. After acquiring the lock, we can then access or read the global variable and do whatever we want to do.

We are doing this because we want to prevent the global variable to be accessed from another thread at the same time it is accessed by another thread.

After performing reading, we also make sure that we release the lock that we have to acquire for reading purposes.

```
def Reader():

   global x

   print('Reader is Reading!')

   lock.acquire()

   print('Shared Data:', x)

   lock.release()

   print()
```

Define Writer Function

This function is similar to the Reader Function hat we created above where we are accessing the global variable for reading purposes in a synchronous way.

Now for writing purposes, we have to do the same approach as for the reader to do.

First, we are going to acquire the lock and then access the global variable and do modification and then after writing, we have to release the lock so that other thread can access the variable.

```
def Writer():
    global x
    print('Writer is Writing!')
    lock.acquire()      #Acquire the lock before Writing
    x += 1              #Write on the shared memory
    print('Writer is Releasing the lock!')
    lock.release()      #Release the lock after Writing
    print()
```

Define Main Condition

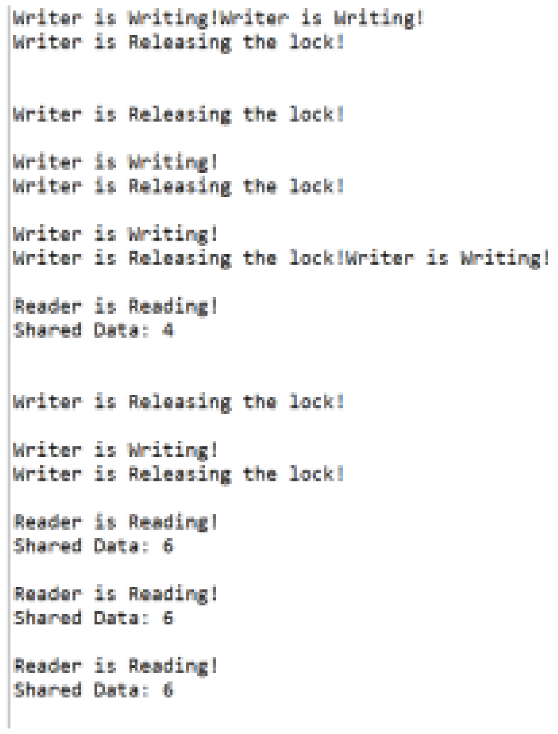
Now, we have created the Reader-Writer Function for accessing the global variable in a synchronous way. Let's define the main condition of the program where we are going to create two threads which try to access these reader and writer function in a synchronous way.

```
if __name__ == '__main__':
    for i in range(0, 10):
        randomNumber = random.randint(0, 100)   #Generate a Random number between 0 to 100
        if(randomNumber > 50):
            Thread1 = thread.Thread(target = Reader)
            Thread1.start()
        else:
            Thread2 = thread.Thread(target = Writer)
            Thread2.start()

Thread1.join()
Thread2.join()
```

***Program* :**

```
import threading as thread
import random

global x                                                #Shared Data
x = 0
lock = thread.Lock()                                    #Lock for synchronising access

def Reader():
   global x
   print('Reader is Reading!')
   lock.acquire()                              #Acquire the lock before Reading (mutex approach)
   print('Shared Data:', x)
   lock.release()                                       #Release the lock after Reading
   print()

def Writer():
   global x
   print('Writer is Writing!')
   lock.acquire()                                       #Acquire the lock before Writing
   x += 1                                               #Write on the shared memory
   print('Writer is Releasing the lock!')
   lock.release()                                       #Release the lock after Writing
   print()

if __name__ == '__main__':
   for i in range(0, 10):
      randomNumber = random.randint(0, 100)       #Generate a Random number between 0 to
100
      if(randomNumber > 50):
         Thread1 = thread.Thread(target = Reader)
         Thread1.start()
      else:
        Thread2 = thread.Thread(target = Writer)
        Thread2.start()

Thread1.join()
Thread2.join()

# print(x)
```

Output **:**



```
Writer is Writing!Writer is Writing!
Writer is Releasing the lock!


Writer is Releasing the lock!

Writer is Writing!
Writer is Releasing the lock!

Writer is Writing!
Writer is Releasing the lock!Writer is Writing!

Reader is Reading!
Shared Data: 4


Writer is Releasing the lock!

Writer is Writing!
Writer is Releasing the lock!

Reader is Reading!
Shared Data: 6

Reader is Reading!
Shared Data: 6

Reader is Reading!
Shared Data: 6
```

RESULT **:**

The usage and functioning of system calls in an operating system and network programming in linux has been studied and familiarized successfully.