CET-255

Operating Systems

Experiment # 10

Experiment Title

- Implementation of Semaphore Mechanism.
- Solving producer-consumer *(Classical Problem)* problem in Python using semaphores.

Assessment of CLO(s): 04

Performed on _____

Student Name:			
Roll No.		**Group**	
Semester		**Session**	

Total (Max)	**Performance (03)**	**Viva (03)**	**File (04)**	**Total (10)**
Marks Obtained				
Remarks (if any)				

Experiment evaluated by

Instructor's Name	**Engr. Bushra Aziz**		
Date		**Signature**	

Semaphores

Semaphore is a simply a variable. This variable is used to solve critical section problem and to achieve process synchronization in the multi-processing environment.
The two most common kinds of semaphores are counting semaphores and binary semaphores. Counting semaphore can take non-negative integer values and Binary semaphore can take the value 0 & 1 only.
A semaphore can only be accessed using the following operations: wait() and signal().

```
wait(Semaphore s) {
while (s==0);     /*
wait until s>0  */
s=s-1;             }
signal(Semaphore s){
s=s+1;
}
```

[In python, acquire() and release() provide wait() and signal() functionality, respectively]

Python's Simple Lock

using class threading.Lock

A simple mutual exclusion lock used to limit access to one thread. This is a semaphore with s = 1.

acquire()

Obtain a lock. The process is blocked until the lock is available.

release()

Release the lock and if another thread is waiting for the lock, wake up that thread.

Python's Semaphore

using class threading.Semaphore(s)

acquire()

Obtain a semaphore. The process is blocked until the semaphore is available.

release()

Release the semaphore and if another thread is waiting for the semaphore, wake up that thread.

Python: Producer-Consumer Solution using Semaphores

The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1. The empty and full semaphores count the number of empty and full buffers. The semaphore empty is initialized to the value max_c (max_c=10 in this example); the semaphore full is initialized to the value 0.

CODE:

```
from threading import Semaphore
import time
import threading
import random
max_c=10
full = Semaphore(0)
empty =Semaphore(max_c)
mutex=Semaphore(1)
queue = []

def producer(i):
  empty.acquire()
  mutex.acquire()
  print("Produced:",i)
  queue.append(i)
  mutex.release()
  full.release()

def consumer(i):
  full.acquire()
  mutex.acquire()
  print("consumer has consumed:",queue.pop(0))
  mutex.release()
  empty.release()


total = []
for i in range(max_c):
 tp=threading.Thread(target=producer,args=[i])
 tc=threading.Thread(target=consumer,args=[i])
 total.append(tp)
 total.append(tc)

random.shuffle(total)

for i in total:
  i.start()
  time.sleep(1)
```

Exercise:

Write a python program that demonstrates the synchronization of Readers and Writer Problem using Semaphore.