DIVYANSHU MAKHARIA
221070036

ASSIGNMENT 3

AIM:

Implement semaphore for the process synchronization.

THEORY:

Semaphores are just normal variables used to coordinate the activities of multiple processes in a computer system. They are used to enforce mutual exclusion, avoid race conditions, and implement synchronization between processes.

The process of using Semaphores provides two operations: wait (P) and signal (V). The wait operation decrements the value of the semaphore, and the signal operation increments the value of the semaphore. When the value of the semaphore is zero, any process that performs a wait operation will be blocked until another process performs a signal operation.

Semaphores are used to implement critical sections, which are regions of code that must be executed by only one process at a time. By using semaphores, processes can coordinate access to shared resources, such as shared memory or I/O devices.

Semaphores are of two types:

1. Binary Semaphore –

   This is also known as a mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problems with multiple processes.

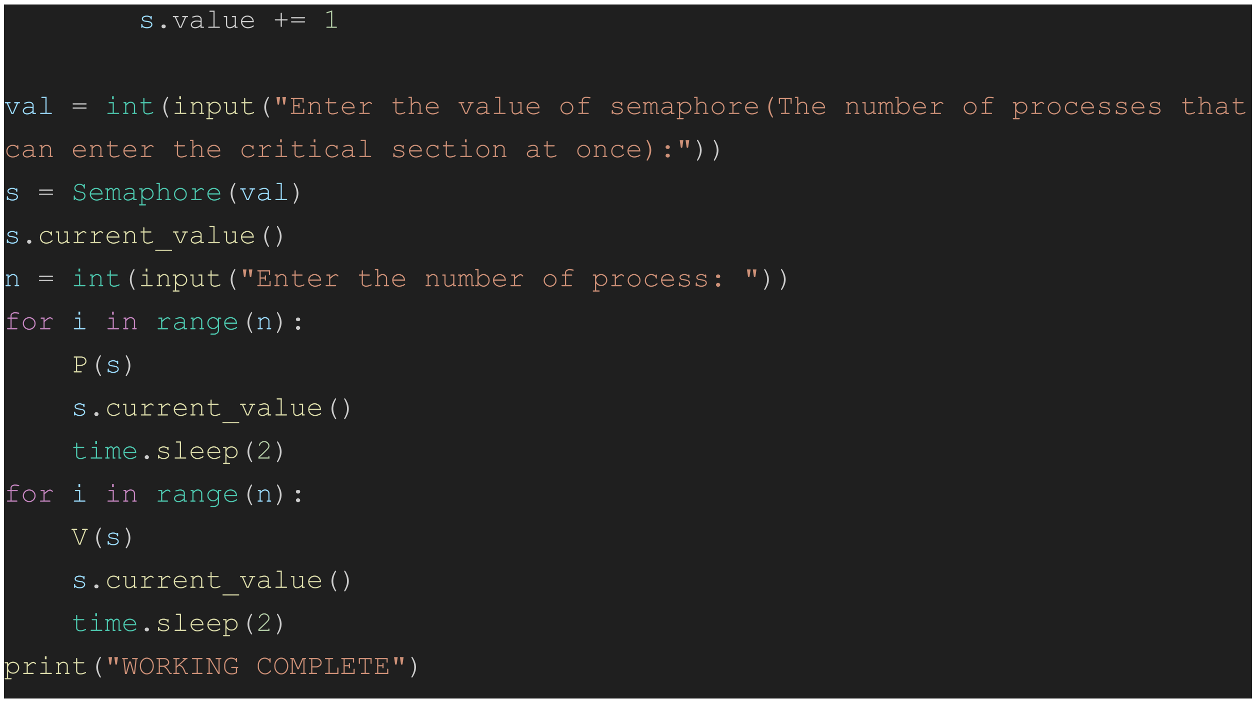
2. Counting Semaphore –

   Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

CODE:

```
import time
class Semaphore:
    def __init__(self,value):
        self.value = value
        self.waitingqueue = []
        self.in_CS = []
    def current_value(self):
        print("Number of process that can enter the critical
section:",self.value)
processno = 1
def P(s):
    global processno
    if s.value == 0:
        print(f"Process {processno} Blocked")
        s.waitingqueue.append(processno)
    else:
        print(f"Process {processno} entered the critical section")
        s.in_CS.append(processno)
        s.value -= 1
    processno+=1
def V(s):
    global processno
    if len(s.in_CS) == 0:
        print("Critical Section Free")
        return
    print(f"Process {s.in_CS.pop(0)} waked up and is completed.")
    if len(s.waitingqueue) != 0:
        tmp = s.waitingqueue.pop(0)
        s.in_CS.append(tmp)
        print(f"Process {tmp} is entering critical section")
    else:
```

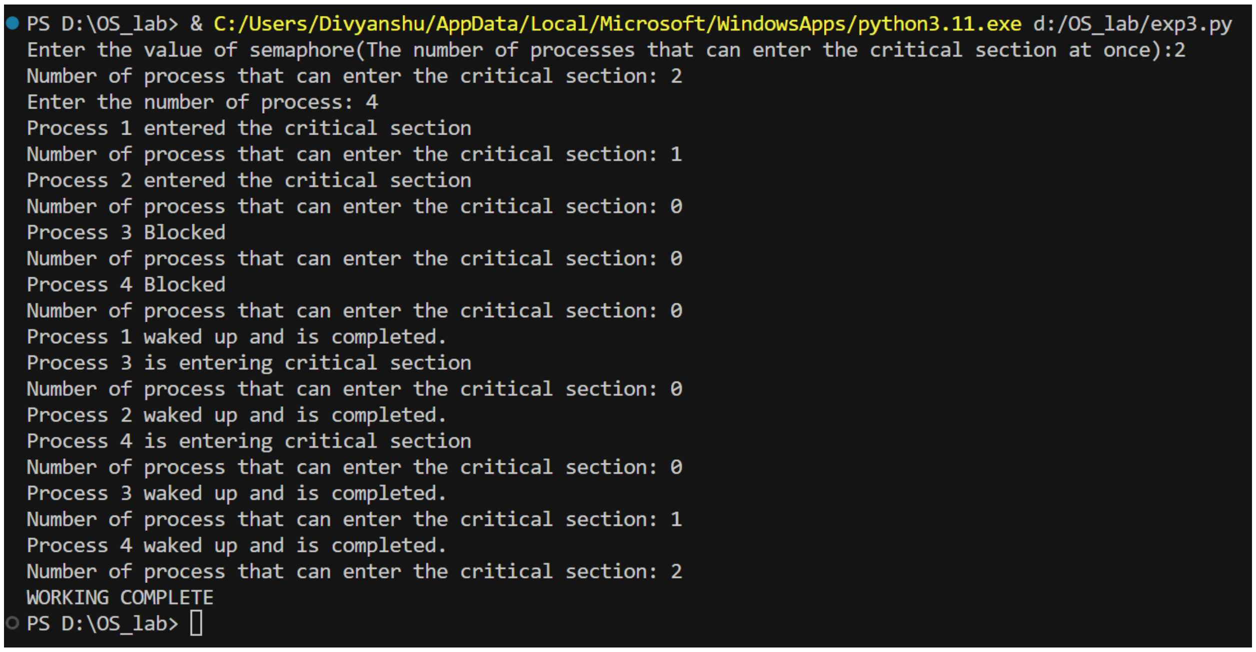
DIVYANSHU MAKHARIA
221070036



```python
        s.value += 1

val = int(input("Enter the value of semaphore(The number of processes that
can enter the critical section at once):"))
s = Semaphore(val)
s.current_value()
n = int(input("Enter the number of process: "))
for i in range(n):
    P(s)
    s.current_value()
    time.sleep(2)
for i in range(n):
    V(s)
    s.current_value()
    time.sleep(2)
print("WORKING COMPLETE")
```

OUTPUT:



```
PS D:\OS_lab> & C:/Users/Divyanshu/AppData/Local/Microsoft/WindowsApps/python3.11.exe d:/OS_lab/exp3.py
Enter the value of semaphore(The number of processes that can enter the critical section at once):2
Number of process that can enter the critical section: 2
Enter the number of process: 4
Process 1 entered the critical section
Number of process that can enter the critical section: 1
Process 2 entered the critical section
Number of process that can enter the critical section: 0
Process 3 Blocked
Number of process that can enter the critical section: 0
Process 4 Blocked
Number of process that can enter the critical section: 0
Process 1 waked up and is completed.
Process 3 is entering critical section
Number of process that can enter the critical section: 0
Process 2 waked up and is completed.
Process 4 is entering critical section
Number of process that can enter the critical section: 0
Process 3 waked up and is completed.
Number of process that can enter the critical section: 1
Process 4 waked up and is completed.
Number of process that can enter the critical section: 2
WORKING COMPLETE
PS D:\OS_lab> 
```

DIVYANSHU MAKHARIA
221070036

CONCLUSION:

Thus,we have successfully learnt about the concept of Semaphore and implemented it using python code.Here,we can understand how processes enter and exit the critical section.