# OS Assignment 8
# Name : Veeransh Shah
# Reg Id: 221070063

**Aim:**
Implementation page replacement algorithm (FIFO,LRU,LFU)

**Theory:**

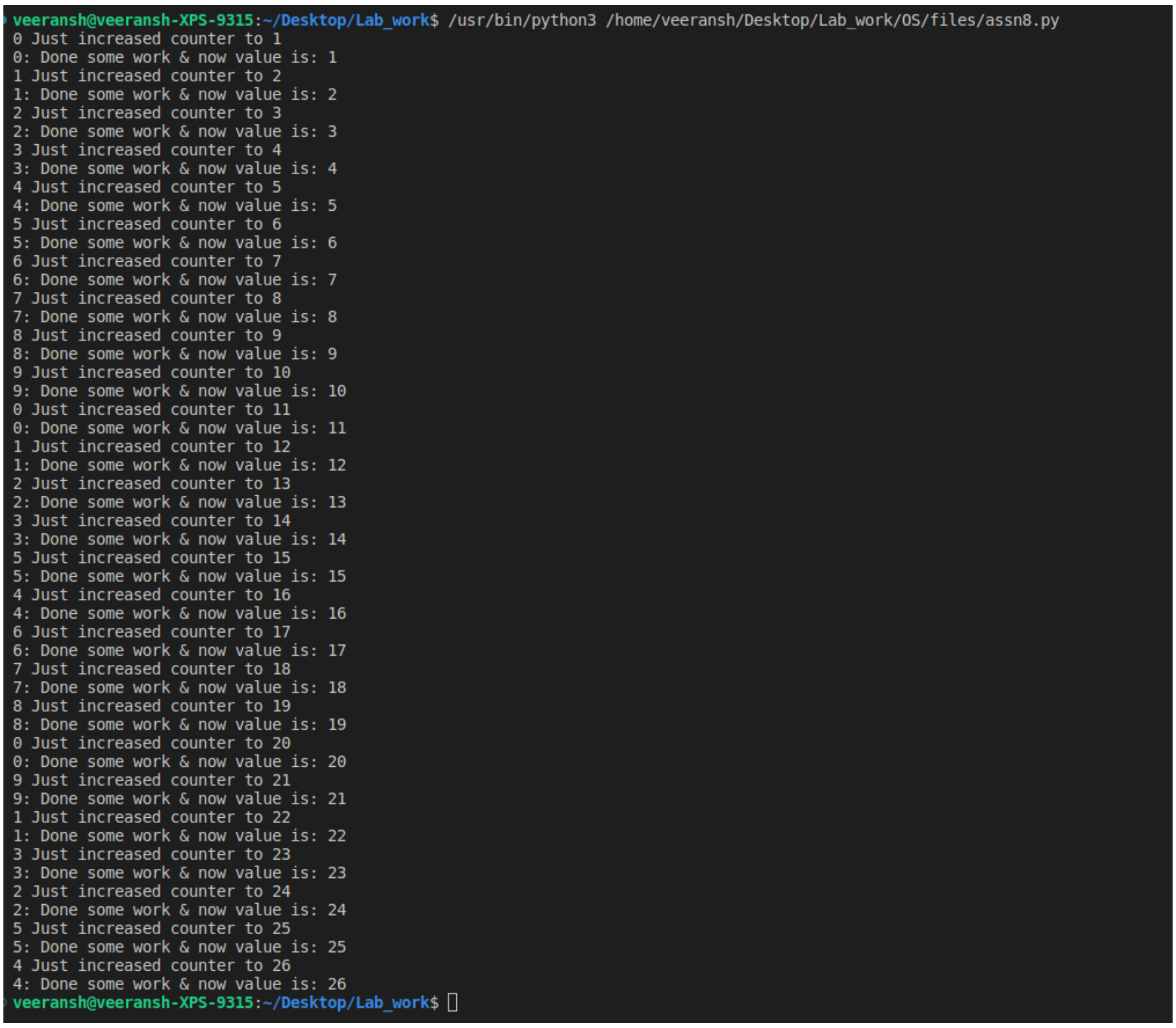
The provided code defines a ThreadCounter class that implements a thread-safe counter using a lock. Each thread executes the count method, which increments the counter until it reaches 20, while ensuring thread safety by acquiring and releasing a lock. After incrementing the counter, the thread prints its number along with the updated counter value. Additionally, the threads simulate work by sleeping for random intervals between 1 and 3 seconds after each increment.

In the main loop, 10 threads are created and initialized with the count method. These threads operate concurrently, incrementing the counter in a thread-safe manner until it reaches 20. The use of locks ensures that only one thread can access and modify the counter at a time, preventing race conditions. By simulating work with random sleep intervals, the code reflects a realistic scenario where threads perform various tasks asynchronously. Overall, this implementation demonstrates effective synchronization techniques using locks to maintain thread safety in a concurrent environment.

**Code:**
```
import time
import threading
import random

class ThreadCounter:
   def __init__(self):
       self.counter = 0
       self.lock = threading.Lock()

   def count(self, thread_no):
       while self.counter < 20:
           self.lock.acquire()
           self.counter += 1
           print(f"{thread_no} Just increased counter to
{self.counter}")
           time.sleep(1)
           print(f"{thread_no}: Done some work & now value is:
{self.counter}")
```

```python
            self.lock.release()
            time.sleep(random.randint(1,3))

tc = ThreadCounter()
for i in range(10):
    t = threading.Thread(target=tc.count, args=(i,))
    t.start()


# Thread counter class : represents a thread safe counter with
attributes 'counter' and lock
# count method : each thread increaments the counter until it reaches
20, ensuring thread safety by using locks, it the prints the thread
number
# threading : main loop creates 10 threads, each initialized with the
count method
# Increament counter : threads acquire lock, increament counter, and
release lock to ensure exclusive access
# simulation : threads simulate work by sleeping for random intervals
after each increament
```

**Output:**



```
veeransh@veeransh-XPS-9315:~/Desktop/Lab_work$ /usr/bin/python3 /home/veeransh/Desktop/Lab_work/OS/files/assn8.py
0 Just increased counter to 1
0: Done some work & now value is: 1
1 Just increased counter to 2
1: Done some work & now value is: 2
2 Just increased counter to 3
2: Done some work & now value is: 3
3 Just increased counter to 4
3: Done some work & now value is: 4
4 Just increased counter to 5
4: Done some work & now value is: 5
5 Just increased counter to 6
5: Done some work & now value is: 6
6 Just increased counter to 7
6: Done some work & now value is: 7
7 Just increased counter to 8
7: Done some work & now value is: 8
8 Just increased counter to 9
8: Done some work & now value is: 9
9 Just increased counter to 10
9: Done some work & now value is: 10
0 Just increased counter to 11
0: Done some work & now value is: 11
1 Just increased counter to 12
1: Done some work & now value is: 12
2 Just increased counter to 13
2: Done some work & now value is: 13
3 Just increased counter to 14
3: Done some work & now value is: 14
5 Just increased counter to 15
5: Done some work & now value is: 15
4 Just increased counter to 16
4: Done some work & now value is: 16
6 Just increased counter to 17
6: Done some work & now value is: 17
7 Just increased counter to 18
7: Done some work & now value is: 18
8 Just increased counter to 19
8: Done some work & now value is: 19
0 Just increased counter to 20
0: Done some work & now value is: 20
9 Just increased counter to 21
9: Done some work & now value is: 21
1 Just increased counter to 22
1: Done some work & now value is: 22
3 Just increased counter to 23
3: Done some work & now value is: 23
2 Just increased counter to 24
2: Done some work & now value is: 24
5 Just increased counter to 25
5: Done some work & now value is: 25
4 Just increased counter to 26
4: Done some work & now value is: 26
veeransh@veeransh-XPS-9315:~/Desktop/Lab_work$
```

**Conclusion:**

In conclusion, the provided code demonstrates the implementation of a thread-safe counter using locks in Python's threading module. By encapsulating the counter and lock within a class, concurrent access to the counter is managed securely, preventing data corruption or race conditions. Through the use of threading, multiple threads can increment the counter simultaneously while ensuring mutual exclusion with the lock mechanism. Moreover, the simulation of work with random sleep intervals adds realism to the concurrent execution scenario. Overall, this code illustrates the importance of synchronization techniques, such as locks, in maintaining thread safety and preventing concurrency issues in multithreaded Python applications.