DIVYANSHU MAKHARIA
221070036

ASSIGNMENT 7

AIM:

Thread synchronization using lock mechanism.

THEORY:

Thread synchronization using lock mechanism is a fundamental technique in multithreading programming to coordinate access to shared resources among multiple threads. In a multi-threaded environment, threads often execute concurrently and may access shared data simultaneously, leading to race conditions, data corruption, or inconsistent program behavior. Locks provide a simple yet powerful mechanism to ensure that only one thread can access a critical section of code, known as a mutual exclusion (mutex), at any given time.

Here's a brief overview of how thread synchronization using locks works:

1. **Acquiring Locks:** Threads that need to access a shared resource first acquire a lock associated with that resource. If the lock is available (i.e., not held by another thread), the thread acquires the lock and proceeds to access the resource.
2. **Mutual Exclusion:** Once a thread acquires a lock, it gains exclusive access to the critical section of code where the shared resource is accessed. Other threads attempting to acquire the same lock will be blocked until the lock is released by the owning thread.
3. **Releasing Locks:** After completing the operations on the shared resource, the thread releases the lock, allowing other waiting threads to acquire it and access the resource.
4. **Preventing Race Conditions:** By ensuring that only one thread can execute the critical section at a time, locks

prevent race conditions and guarantee the consistency of shared data.

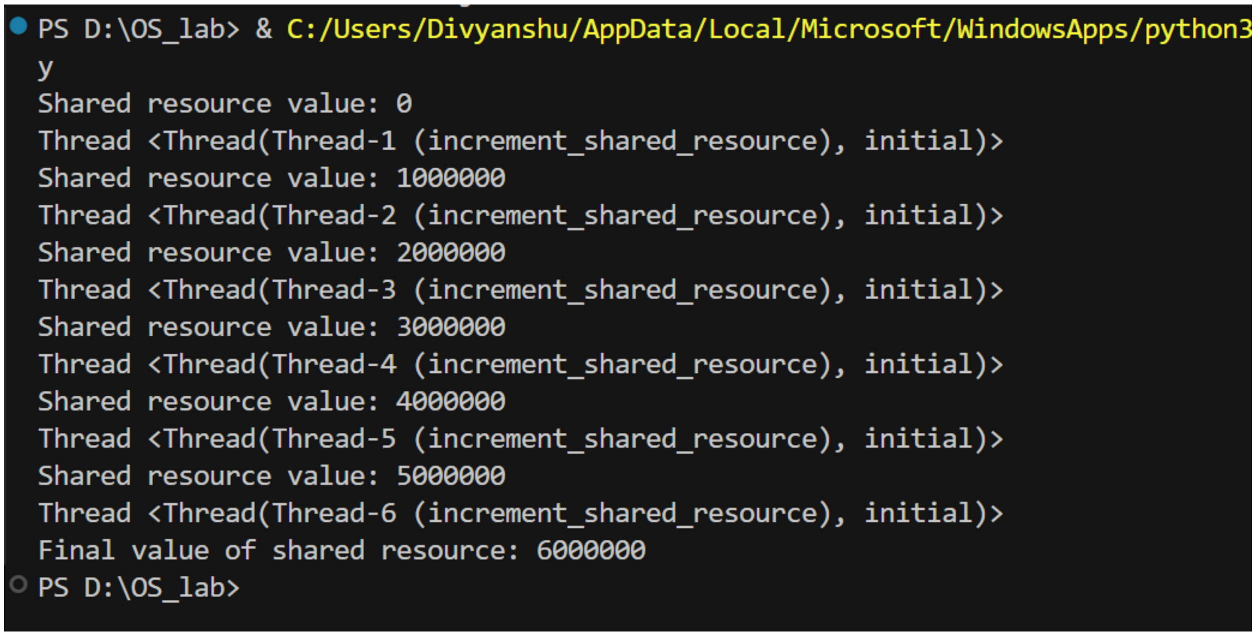
5. **Deadlock Avoidance:** Care must be taken to avoid deadlock situations where two or more threads are waiting indefinitely for locks held by each other. Techniques such as lock ordering and timeout mechanisms can help prevent deadlocks.

CODE:

```
import threading
import time

shared_resource = 0

lock = threading.Lock()

def increment_shared_resource():
    global shared_resource
    for _ in range(1000000):
        lock.acquire()
        shared_resource += 1
        # time.sleep(1)
        lock.release()

# Create multiple threads
threads = []
for _ in range(6):
    thread = threading.Thread(target=increment_shared_resource)
    threads.append(thread)
    print("Shared resource value:",shared_resource)
    print("Thread",thread)
    thread.start()
    time.sleep(1)
```

```python
# Wait for all threads to complete
for thread in threads:
    thread.join()

# Print the final value of the shared resource
print("Final value of shared resource:", shared_resource)
```

OUTPUT:



```
PS D:\OS_lab> & C:/Users/Divyanshu/AppData/Local/Microsoft/WindowsApps/python3
y
Shared resource value: 0
Thread <Thread(Thread-1 (increment_shared_resource), initial)>
Shared resource value: 1000000
Thread <Thread(Thread-2 (increment_shared_resource), initial)>
Shared resource value: 2000000
Thread <Thread(Thread-3 (increment_shared_resource), initial)>
Shared resource value: 3000000
Thread <Thread(Thread-4 (increment_shared_resource), initial)>
Shared resource value: 4000000
Thread <Thread(Thread-5 (increment_shared_resource), initial)>
Shared resource value: 5000000
Thread <Thread(Thread-6 (increment_shared_resource), initial)>
Final value of shared resource: 6000000
PS D:\OS_lab>
```

CONCLUSION:

Thus,we have learnt about the concept of threading.Thread synchronization using locks is a fundamental concept in multithreading to ensure that only one thread can access a shared resource at a time, preventing data corruption or race conditions.