```python
import threading
import time
import random

# Shared resource (e.g., a counter)
counter = 0

# Lock to prevent race conditions
lock = threading.Lock()

# Semaphore to allow a limited number of threads to access a resource at a
time
semaphore = threading.Semaphore(3)  # Only 3 threads can access the
resource at once

def increment_counter(thread_id):
    global counter

    # Acquiring the semaphore (limits concurrent access)
    with semaphore:
        print(f"Thread-{thread_id} is waiting for access...")
        time.sleep(random.uniform(0.5, 1.5))  # Simulate work

        # Acquiring lock to modify shared resource safely
        with lock:
            print(f"Thread-{thread_id} has acquired the lock.")
            temp = counter
            time.sleep(random.uniform(0.1, 0.5))  # Simulate processing
delay
            counter = temp + 1
            print(f"Thread-{thread_id} updated counter to {counter}")

        print(f"Thread-{thread_id} has released the lock.")

# Creating multiple threads
threads = []
num_threads = 10  # Number of threads

for i in range(num_threads):
    t = threading.Thread(target=increment_counter, args=(i,))
    threads.append(t)
    t.start()

# Wait for all threads to complete
for t in threads:
```

```
    t.join()

print(f"\nFinal counter value: {counter}")
```