Practical No 6

Aim: Readers-Writers Problem- Synchronization in Shared Access

6.1. Implement Access reader and writer prioritization.
6.2. Use semaphores to allow multiple readers or exclusive.

```
import threading
import time
import random

# Semaphores
mutex = threading.Semaphore(1)  # Protects readerCount
db = threading.Semaphore(1)     # Controls access to sharedData

# Shared resource and reader count
sharedData = 0
readerCount = 0
isRunning = True  # Stop flag

def reader(readerId):
    global readerCount, sharedData, isRunning
    while isRunning:
        # Entry section
        mutex.acquire()
        readerCount += 1
        if readerCount == 1:
            db.acquire()  # First reader locks db
        mutex.release()

        # Critical section
        print(f"[Reader] {readerId} reads {sharedData}")
        time.sleep(random.uniform(0.2, 0.5))

        # Exit section
        mutex.acquire()
        readerCount -= 1
        if readerCount == 0:
            db.release()  # Last reader unlocks db
        mutex.release()

        # Pause before next read
        time.sleep(random.uniform(0.5, 1.0))

def writer(writerId):
    global sharedData, isRunning
    while isRunning:
        db.acquire()  # Exclusive access
        sharedData += 1
        print(f"[Writer] {writerId} writes {sharedData}")
        time.sleep(random.uniform(0.3, 0.6))
        db.release()
```

```
        # Pause before next write
        time.sleep(random.uniform(0.8, 1.5))

if __name__ == "__main__":
    readers = [threading.Thread(target=reader, args=(i,)) for i in range(3)]
    writers = [threading.Thread(target=writer, args=(i,)) for i in range(2)]

    for t in readers + writers:
        t.start()

    time.sleep(10)  # Run simulation for 10 sec
    isRunning = False  # Stop all threads

    for t in readers + writers:
        t.join()

    print("Simulation finished.")
```

```
[Reader] 0 reads 0
[Reader] 1 reads 0
[Reader] 2 reads 0
[Writer] 0 writes 1
[Writer] 1 writes 2
[Reader] 2 reads 2[Reader] 1 reads 2

[Reader] 0 reads 2
[Writer] 0 writes 3
[Reader] 0 reads 3[Reader] 2 reads 3

[Reader] 1 reads 3
[Writer] 1 writes 4
[Writer] 0 writes 5
[Reader] 1 reads 5[Reader] 0 reads 5[Reader] 2 reads 5



[Writer] 1 writes 6
[Writer] 0 writes 7
[Reader] 1 reads 7
[Reader] 2 reads 7
[Reader] 0 reads 7
[Writer] 1 writes 8
[Reader] 0 reads 8
[Reader] 1 reads 8
[Reader] 2 reads 8
[Writer] 0 writes 9
[Reader] 0 reads 9
[Reader] 2 reads 9
[Reader] 1 reads 9
[Writer] 1 writes 10
[Writer] 0 writes 11
[Reader] 0 reads 11[Reader] 1 reads 11[Reader] 2 reads 11



[Writer] 1 writes 12
Simulation finished.
```

6.3. Extend to fairness writer access. in access and deadlock prevention.

```
import threading
import time
import random

# Semaphores
mutex = threading.Semaphore(1)        # Protects readerCount
db = threading.Semaphore(1)           # Controls access to sharedData
serviceQueue = threading.Semaphore(1) # Fairness: queue for both readers/writers

# Shared resource and counters
sharedData = 0
readerCount = 0
isRunning = True  # Stop flag

def reader(readerId):
    global readerCount, sharedData, isRunning
    while isRunning:
        # Fairness: wait in queue
        serviceQueue.acquire()
        mutex.acquire()
        readerCount += 1
        if readerCount == 1:
            db.acquire()  # First reader locks db
        mutex.release()
        serviceQueue.release()

        # Critical section
        print(f"[Reader] {readerId} reads {sharedData}")
        time.sleep(random.uniform(0.2, 0.5))

        # Exit section
        mutex.acquire()
        readerCount -= 1
        if readerCount == 0:
            db.release()  # Last reader unlocks db
        mutex.release()

        # Pause before next read
        time.sleep(random.uniform(0.5, 1.0))

def writer(writerId):
    global sharedData, isRunning
    while isRunning:
        # Fairness: wait in queue
        serviceQueue.acquire()
        db.acquire()  # Exclusive access
        serviceQueue.release()

        # Critical section
        sharedData += 1
        print(f"[Writer] {writerId} writes {sharedData}")
        time.sleep(random.uniform(0.3, 0.6))
```

```
        db.release()

        # Pause before next write
        time.sleep(random.uniform(0.8, 1.5))

if __name__ == "__main__":
    readers = [threading.Thread(target=reader, args=(i,)) for i in range(3)]
    writers = [threading.Thread(target=writer, args=(i,)) for i in range(2)]

    for t in readers + writers:
        t.start()

    time.sleep(10)  # Run simulation for 10 sec
    isRunning = False  # Stop all threads

    for t in readers + writers:
        t.join()

    print("Simulation finished.")
```

```
[Reader] 0 reads 0
[Reader] 1 reads 0
[Reader] 2 reads 0
[Writer] 0 writes 1
[Writer] 1 writes 2
[Reader] 0 reads 2[Reader] 2 reads 2

[Reader] 1 reads 2
[Writer] 0 writes 3
[Reader] 2 reads 3[Reader] 1 reads 3

[Writer] 1 writes 4
[Reader] 0 reads 4
[Reader] 1 reads 4
[Reader] 2 reads 4
[Writer] 0 writes 5
[Writer] 1 writes 6
[Reader] 0 reads 6[Reader] 1 reads 6

[Reader] 2 reads 6
[Writer] 0 writes 7
[Reader] 1 reads 7
[Reader] 2 reads 7
[Reader] 0 reads 7
[Writer] 1 writes 8
[Reader] 0 reads 8
[Writer] 0 writes 9
[Reader] 2 reads 9[Reader] 1 reads 9

[Writer] 1 writes 10
[Reader] 0 reads 10[Reader] 2 reads 10

[Writer] 0 writes 11
[Reader] 1 reads 11
[Writer] 1 writes 12
[Reader] 2 reads 12
[Reader] 0 reads 12
Simulation finished.
```