# **Lab Experiment 04**

**Sushil Shetty**

**22BAI10297**

**Bounded Buffer**

**Code :**

from threading import Semaphore, Thread

class BoundedBuffer:

    def \_\_init\_\_(self, capacity):

        self.buffer = [None] \* capacity

        self.in\_index = 0

        self.out\_index = 0

        self.capacity = capacity

        self.empty = Semaphore(capacity)

        self.full = Semaphore(0)

        self.lock = Semaphore(1)  # Mutex for critical sections

    def produce(self, item):

        self.empty.acquire()

        with self.lock:

            self.buffer[self.in\_index] = item

            self.in\_index = (self.in\_index + 1) % self.capacity

        self.full.release()

    def consume(self):

        self.full.acquire()

        with self.lock:

            item = self.buffer[self.out\_index]

            self.out\_index = (self.out\_index + 1) % self.capacity

        self.empty.release()

        return item

def producer(buffer, items):

    for item in items:

        buffer.produce(item)

        print(f"Produced: {item}")

def consumer(buffer):

    while True:

        item = buffer.consume()

        print(f"Consumed: {item}")

# Example usage

buffer = BoundedBuffer(5)  # Create a buffer with capacity 5

items = [1, 2, 3, 4, 5]

producer\_thread = Thread(target=producer, args=(buffer, items))

consumer\_thread = Thread(target=consumer, args=(buffer,))

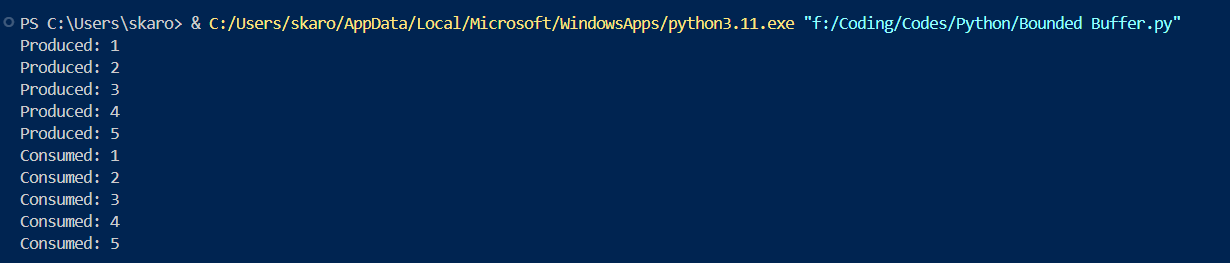
producer\_thread.start()

consumer\_thread.start()

producer\_thread.join()

consumer\_thread.join()

**Output :**

****

**Readers-Writers**

**Code :**

from threading import Semaphore, Thread

class ReadWrite:

    def \_\_init\_\_(self):

        self.read\_count = 0

        self.mutex = Semaphore(1)

        self.wrt = Semaphore(1)

    def reader\_enter(self):

        self.mutex.acquire()

        self.read\_count += 1

        if self.read\_count == 1:

            self.wrt.acquire()

        self.mutex.release()

    def reader\_exit(self):

        self.mutex.acquire()

        self.read\_count -= 1

        if self.read\_count == 0:

            self.wrt.release()

        self.mutex.release()

    def writer\_enter(self):

        self.wrt.acquire()

    def writer\_exit(self):

        self.wrt.release()

def reader(rw):

    rw.reader\_enter()

    # Simulate reading process

    print("Reader is reading...")

    rw.reader\_exit()

def writer(rw):

    rw.writer\_enter()

    # Simulate writing process

    print("Writer is writing...")

    rw.writer\_exit()

# Example usage

rw = ReadWrite()

reader\_threads = [Thread(target=reader, args=(rw,)) for \_ in range(3)]

writer\_thread = Thread(target=writer, args=(rw,))

for thread in reader\_threads:

    thread.start()

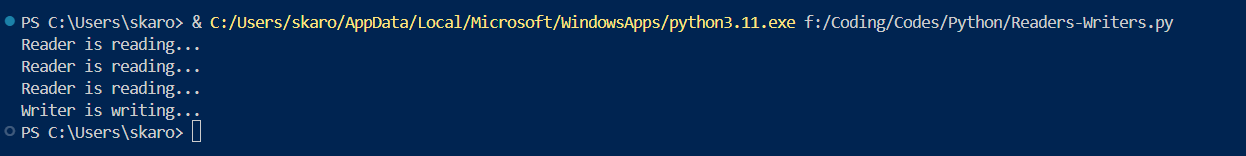
writer\_thread.start()

for thread in reader\_threads:

    thread.join()

writer\_thread.join()

**Output :**

****

**Dining-Philosopher**

**Code :**

from threading import Semaphore, Thread

class DiningPhilosophers:

    def \_\_init\_\_(self, num\_philosophers):

        self.num\_philosophers = num\_philosophers

        self.forks = [Semaphore(1) for \_ in range(num\_philosophers)]  # List of semaphores for forks

    def pick\_up\_forks(self, phil\_number):

        self.forks[phil\_number].acquire()  # Acquire left fork

        print(f"Philosopher {phil\_number + 1} acquires left fork.")

        self.forks[(phil\_number + 1) % self.num\_philosophers].acquire()  # Acquire right fork

        print(f"Philosopher {phil\_number + 1} acquires right fork. Eating...")

    def put\_down\_forks(self, phil\_number):

        self.forks[phil\_number].release()  # Release left fork

        print(f"Philosopher {phil\_number + 1} puts down left fork.")

        self.forks[(phil\_number + 1) % self.num\_philosophers].release()  # Release right fork

        print(f"Philosopher {phil\_number + 1} puts down right fork. Thinking...")

def philosopher(phil\_number, table):

    while True:

        table.pick\_up\_forks(phil\_number)  # Attempt to pick up forks

        # Simulate eating (thinking can be added here)

        table.put\_down\_forks(phil\_number)  # Put down forks after eating

# Example usage

num\_philosophers = 5

table = DiningPhilosophers(num\_philosophers)

philosophers = [Thread(target=philosopher, args=(i, table)) for i in range(num\_philosophers)]

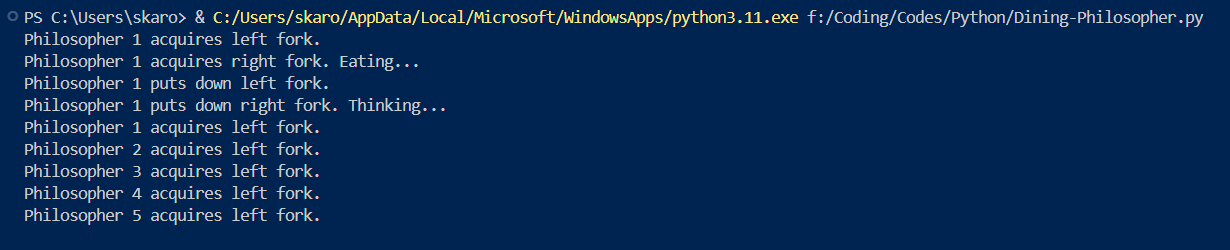
for p in philosophers:

    p.start()

for p in philosophers:

    p.join()

**Output :**

****