Lab Experiment 04

Aromal Kunnel
22BAI10288

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}")
```

```python
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 :

```
PS C:\Users\skaro> & C:/Users/skaro/AppData/Local/Microsoft/WindowsApps/python3.11.exe "f:/Coding/Codes/Python/Bounded Buffer.py"
Produced: 1
Produced: 2
Produced: 3
Produced: 4
Produced: 5
Consumed: 1
Consumed: 2
Consumed: 3
Consumed: 4
Consumed: 5
```

Readers-Writers

Code :

```python
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 :

```
PS C:\Users\skaro> & C:/Users/skaro/AppData/Local/Microsoft/WindowsApps/python3.11.exe f:/Coding/Codes/Python/Readers-Writers.py
Reader is reading...
Reader is reading...
Reader is reading...
Writer is writing...
PS C:\Users\skaro>
```

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
```

```python
# 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 :

```
PS C:\Users\skaro> & C:/Users/skaro/AppData/Local/Microsoft/WindowsApps/python3.11.exe f:/Coding/Codes/Python/Dining-Philosopher.py
Philosopher 1 acquires left fork.
Philosopher 1 acquires right fork. Eating...
Philosopher 1 puts down left fork.
Philosopher 1 puts down right fork. Thinking...
Philosopher 1 acquires left fork.
Philosopher 2 acquires left fork.
Philosopher 3 acquires left fork.
Philosopher 4 acquires left fork.
Philosopher 5 acquires left fork.
```