Formal Methods Lab - Experiment 4 Solutions

1. -Calculus Communication System

Two processes communicate using dynamically created channels. The parent process sends a message to the child, which receives it.

2. CCS Processes with Synchronization

Three CCS processes execute actions in parallel, ensuring synchronization using relabeling and restriction where required.

3. Load Balancer Simulation

A dispatcher assigns tasks to multiple workers, ensuring fairness in task distribution and preventing starvation.

4. Bisimulation Equivalence Check

A Python-based verification system checks whether two finite-state processes exhibit the same external behavior.

5. Producer-Consumer System

A producer generates tasks and places them in a queue while a consumer retrieves and processes them, ensuring synchronization and preventing deadlocks.

```
import multiprocessing
import threading
import queue

# 1. -Calculus Communication System
def pi_process(pipe):
    msg = pipe.recv()
    print("Received message:", msg)
    pipe.close()

parent_conn, child_conn = multiprocessing.Pipe()
proc = multiprocessing.Process(target=pi_process, args=(child_conn,))
proc.start()
parent_conn.send("Hello from -Calculus!")
proc.join()

# 2. CCS Processes with Synchronization
class CCS_Process:
    def __init__(self, name, action):
        self.name = name
        self.action = action

    def execute(self):
        print(f"{self.name} executes {self.action}")

P = CCS_Process("P", "a")
```

```
Q = CCS_Process("Q", "b")
R = CCS_Process("R", "c")
P.execute()
Q.execute()
R.execute()

# 3. Load Balancer Simulation
def worker(task_queue, worker_id):
    while not task_queue.empty():
        try:
            task = task_queue.get_nowait()
            print(f"Worker {worker_id} processing task {task}")
        except queue.Empty:
            break

tasks = queue.Queue()
for i in range(5):
    tasks.put(f"Task-{i}")

workers = []
for i in range(3):
    t = threading.Thread(target=worker, args=(tasks, i))
    workers.append(t)
    t.start()
for t in workers:
    t.join()

# 4. Bisimulation Equivalence Check
def bisimulation(P, Q):
    return P == Q

P_transitions = {"a": "s1", "b": "s2"}
Q_transitions = {"a": "s1", "b": "s2"}
print("Bisimulation equivalent:", bisimulation(P_transitions, Q_transitions))

# 5. Producer-Consumer System
class ProducerConsumer:
    def __init__(self):
        self.queue = queue.Queue()

    def producer(self):
        for i in range(5):
            self.queue.put(f"Item-{i}")
            print(f"Produced Item-{i}")

    def consumer(self):
        while not self.queue.empty():
            item = self.queue.get()
            print(f"Consumed {item}")

pc = ProducerConsumer()
producer_thread = threading.Thread(target=pc.producer)
consumer_thread = threading.Thread(target=pc.consumer)
producer_thread.start()
```

```
producer_thread.join()
consumer_thread.start()
consumer_thread.join()
```