Operating Systems Lab
Fall 2024-25(L59+60)
Student Name:- Parth Suri
Registration No:- 22BDS0116
Class No. :- VL2024250102445
Faculty Name:- Rahul Srivastava

Experiment Title: Implementing the Bakery Algorithm in Python

Lab Title: Mutual Exclusion with the Bakery Algorithm in Python

Objective:
• Understand the Bakery Algorithm for achieving mutual exclusion.
• Implement the algorithm in Python to simulate process synchronization.
• Explore alternatives to busy waiting using Python's threading constructs.
Prerequisites:
• Basic understanding of threading and synchronization in operating systems.
• Familiarity with Python programming and multithreading.

Theory:
The Bakery Algorithm is a software-based solution for mutual exclusion, ensuring that multiple threads (or processes) can safely access a shared resource without conflicts. The algorithm simulates a bakery where each customer (thread) takes a numbered ticket and waits for their turn. The thread with the smallest ticket number gets access to the critical section first.

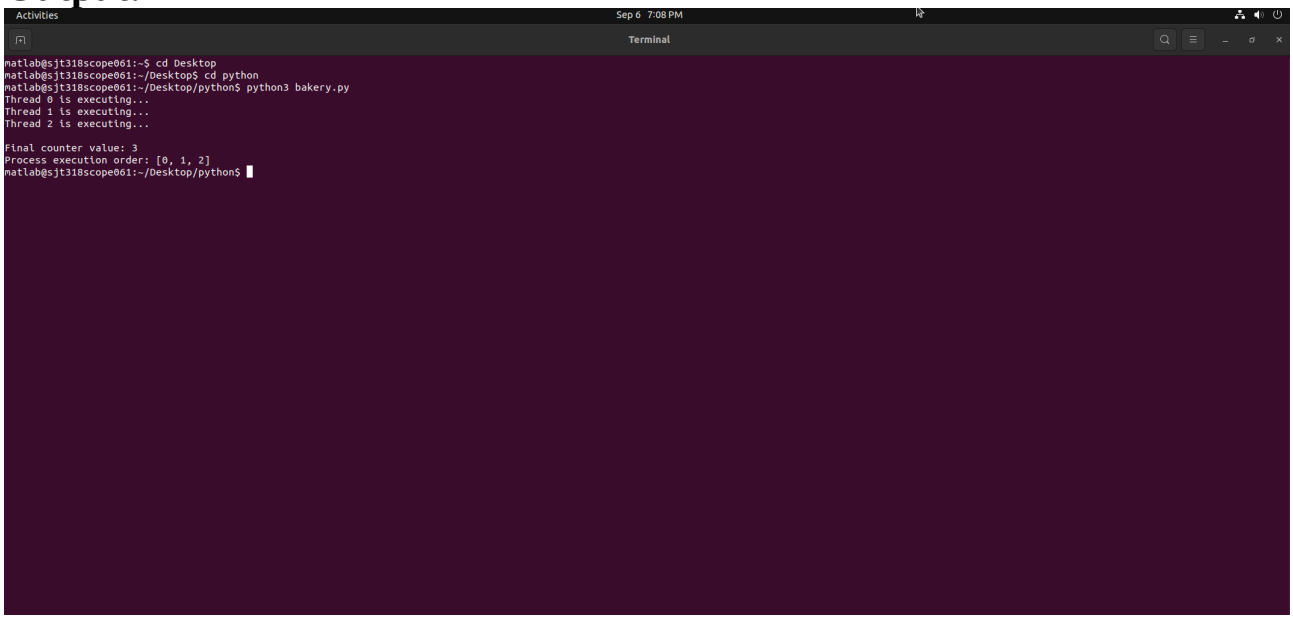
Characteristics of the Bakery Algorithm:
1. Fairness: Every process will eventually get its turn.
2. No Deadlock: Processes will not get stuck indefinitely waiting for each other.
3. Busy Waiting: Threads actively wait by repeatedly checking conditions, which can be inefficient.

Task 1: Implementing the Bakery Algorithm with Busy Waiting Step-by-Step Guide:
1. Step 1: Define the BakeryLock class with flag[] and label[] arrays to manage lock acquisition.
2. Step 2: Implement the lock() and unlock() methods using busy waiting.
3. Step 3: Simulate a critical section using multiple threads and a shared resource (e.g., a counter).
4. Step 4: Track the execution order of the threads.

Code:-

```
import threading

class BakeryLock:
	def __init__(self, n):
		self.n = n
		self.flag= [False] * n
		self.label = [0] * n

	def lock(self, thread_id):
		self.flag[thread_id] = True;
		self.label[thread_id]= max(self.label) + 1
		while any(self.flag[j] and (self.label[j], j) < (self.label[thread_id],
thread_id) for j in range(self.n) if j != thread_id):
			pass

	def unlock(self, thread_id):
		self.flag[thread_id] = False

counter = 0
lock = BakeryLock(3)
execution_order = []

def critical_section(thread_id):
	global counter
	lock.lock(thread_id)
	execution_order.append(thread_id)
	print(f"Thread {thread_id} is executing...")
	counter+=1
	lock.unlock(thread_id)

threads = []
for i in range(3):
	t = threading.Thread(target=critical_section, args=(i,))
	threads.append(t)
	t.start()

for t in threads:
	t.join()

print("\nFinal counter value:", counter)
print("Process execution order:", execution_order)
```

Output:-


Activities
Sep 6 7:08 PM
Terminal
matlab@sjt318scope061:~$ cd Desktop
matlab@sjt318scope061:~/Desktop$ cd python
matlab@sjt318scope061:~/Desktop/python$ python3 bakery.py
Thread 0 is executing...
Thread 1 is executing...
Thread 2 is executing...
Final counter value: 3
Process execution order: [0, 1, 2]
matlab@sjt318scope061:~/Desktop/python$