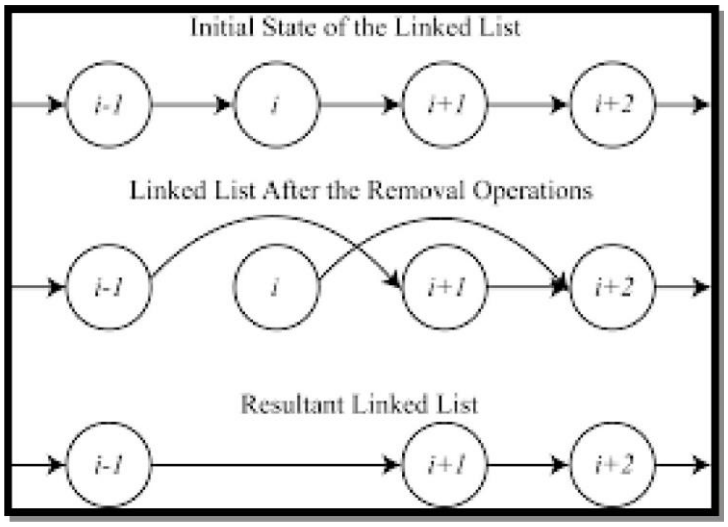
EXPERIMENT NO : 07

AIM: Implementation of a Mutual Exclusion Algorithm.

Theory:

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process cannot enter its critical section while another concurrent process is currently present or executing in its critical section i.e. only one process is allowed to execute the critical section at any given instance of time.

Mutual exclusion in single computer system Vs. distributed system: In single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variable (For example: Semaphores) mutual exclusion problem can be easily solved. In Distributed systems, we neither have shared memory nor a common physical clock and there for we cannot solve mutual exclusion problem using shared variables. To eliminate the mutual exclusion problem in distributed system approach based on message passing is used. A site in distributed system do not have complete information of state of the system due to lack of shared memory and a common physical clock.


Initial State of the Linked List
i-1
i
i+1
i+2
Linked List After the Removal Operations
i-1
i
i+1
i+2
Resultant Linked List
i-1
i+1
i+2

Requirements of Mutual exclusion Algorithm:

- No Deadlock: Two or more site should not endlessly wait for any message that will never arrive.
- No Starvation: Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site is repeatedly executing critical section

- Fairness: Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made i.e. Critical section execution requests should be executed in the order of their arrival in the system.
- Fault Tolerance: In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Some points are need to be taken in consideration to understand mutual exclusion fully •

l) It is an issue/problem which frequently arises when concurrent access to shared resources by several sites is involved.

2) It is a fundamental issue in the design of distributed systems.

3) Mutual exclusion for a single computer is not applicable for the shared resources since it involves resource distribution, transmission delays, and lack of global information.

Solution to distributed mutual exclusion: As we know shared variables or a local kernel cannot be used to implement mutual exclusion in distributed systems. Message passing is a way to implement mutual exclusion. Below are the three approaches based on message passing to implement mutual exclusion in distributed systems:

1. **Token Based Algorithm:**

- A unique token is shared among all the sites.
- If a site possesses the unique token, it is allowed to enter its critical section
- This approach uses sequence number to order requests for the critical section.
- Each request for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.
- This approach ensures Mutual exclusion as the token is unique

Example : Suzuki—Kasami Algorithm

2. **Non-token based approach:**

- o A site communicates with other sites in order to determine which sites should execute critical section next. This requires exchange of two or more successive round of messages among sites.
- o This approach uses timestamps instead of sequence number to order requests for the critical section.
- o Whenever a site make request for critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests.
- o All algorithm which follows non-token based approach maintains a logical clock. Logical clocks get updated according to Lamport's scheme

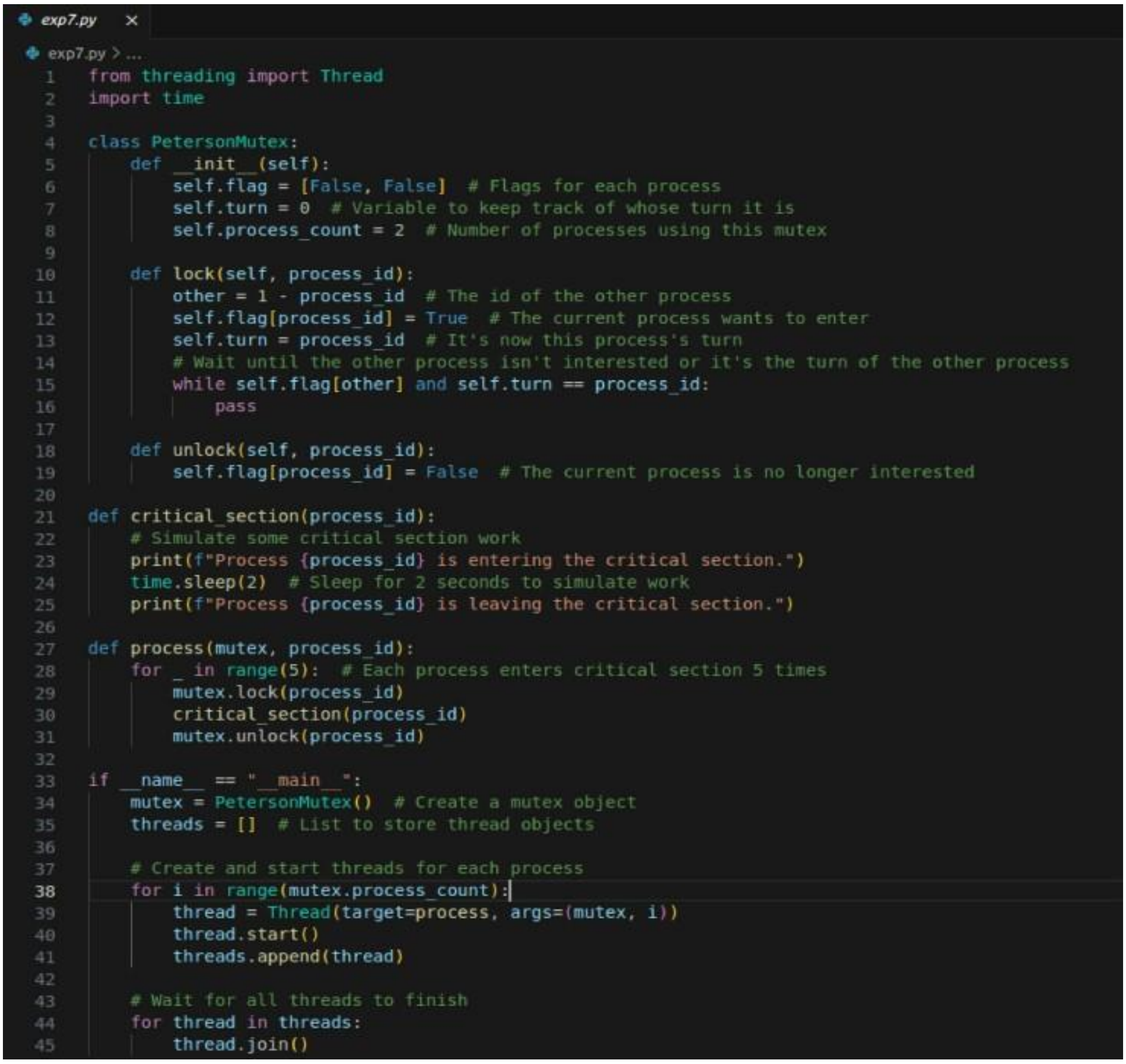
Example : Ricart-Agrawala Algorithm

3. **Quorum based approach:**

- Instead of requesting permission to execute the critical section from all other sites, Each site requests only a subset of sites which is called a quorum.
- Any two subsets of sites or Quorum contains a common site.
- This common site is responsible to ensure mutual exclusion

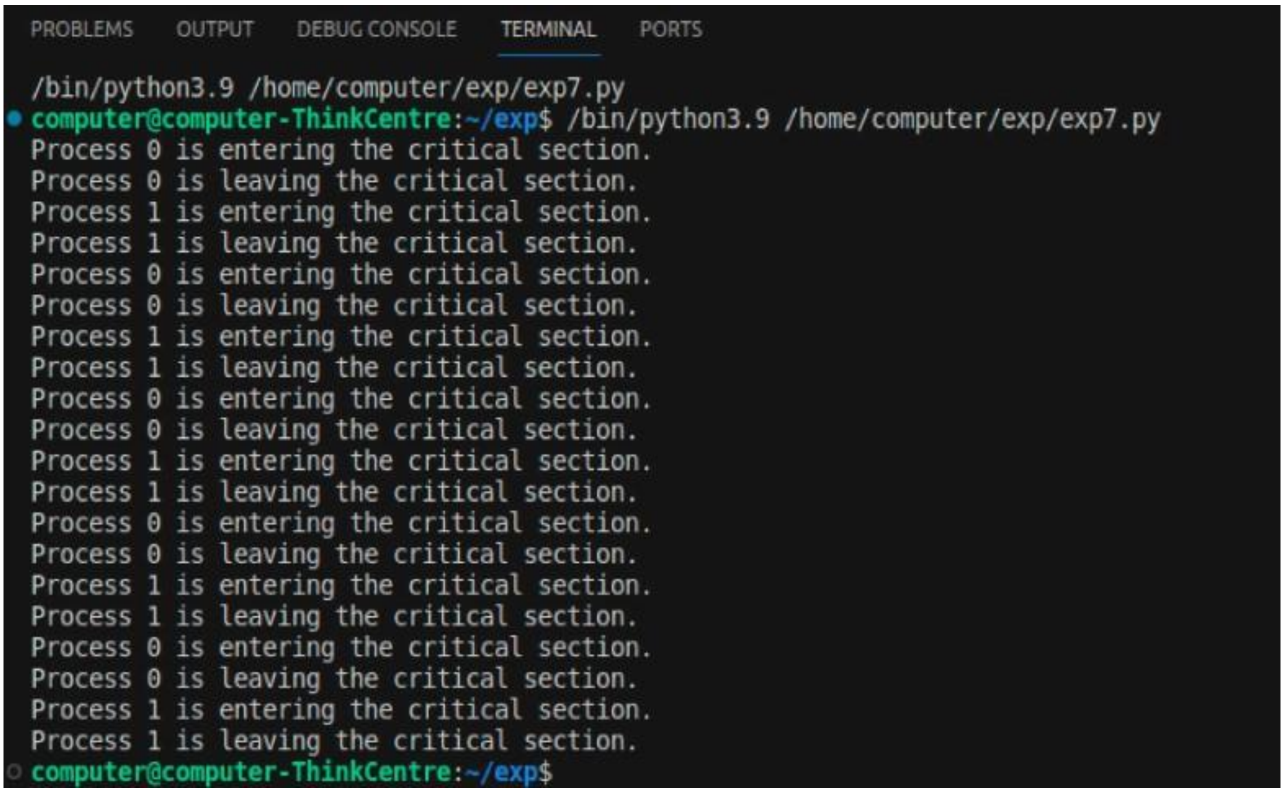
Example : Maekawa's Algorithm

Code :



```
exp7.py  ×
exp7.py > ...
from threading import Thread
import time

class PetersonMutex:
    def __init__(self):
        self.flag = [False, False]  # Flags for each process
        self.turn = 0  # Variable to keep track of whose turn it is
        self.process_count = 2  # Number of processes using this mutex

    def lock(self, process_id):
        other = 1 - process_id  # The id of the other process
        self.flag[process_id] = True  # The current process wants to enter
        self.turn = process_id  # It's now this process's turn
        # Wait until the other process isn't interested or it's the turn of the other process
        while self.flag[other] and self.turn == process_id:
            pass

    def unlock(self, process_id):
        self.flag[process_id] = False  # The current process is no longer interested

def critical_section(process_id):
    # Simulate some critical section work
    print(f"Process {process_id} is entering the critical section.")
    time.sleep(2)  # Sleep for 2 seconds to simulate work
    print(f"Process {process_id} is leaving the critical section.")

def process(mutex, process_id):
    for _ in range(5):  # Each process enters critical section 5 times
        mutex.lock(process_id)
        critical_section(process_id)
        mutex.unlock(process_id)

if __name__ == "__main__":
    mutex = PetersonMutex()  # Create a mutex object
    threads = []  # List to store thread objects

    # Create and start threads for each process
    for i in range(mutex.process_count):
        thread = Thread(target=process, args=(mutex, i))
        thread.start()
        threads.append(thread)

    # Wait for all threads to finish
    for thread in threads:
        thread.join()
```

Output:



```
/bin/python3.9 /home/computer/exp/exp7.py
computer@computer-ThinkCentre:~/exp$ /bin/python3.9 /home/computer/exp/exp7.py
Process 0 is entering the critical section.
Process 0 is leaving the critical section.
Process 1 is entering the critical section.
Process 1 is leaving the critical section.
Process 0 is entering the critical section.
Process 0 is leaving the critical section.
Process 1 is entering the critical section.
Process 1 is leaving the critical section.
Process 0 is entering the critical section.
Process 0 is leaving the critical section.
Process 1 is entering the critical section.
Process 1 is leaving the critical section.
Process 0 is entering the critical section.
Process 0 is leaving the critical section.
Process 1 is entering the critical section.
Process 1 is leaving the critical section.
Process 0 is entering the critical section.
Process 0 is leaving the critical section.
Process 1 is entering the critical section.
Process 1 is leaving the critical section.
computer@computer-ThinkCentre:~/exp$
```

Conclusion:
Thus, we had Successfully Implemented Mutual Exclusion Algorithm.