St. Francis Institute of Technology

Department of Computer Engineering

Academic Year: 2021-2022 **Semester: VIII**

Subject: DCL **Class/Branch/: BE/CMPNA**

Name :- Nithin Menezes **Roll Number: 56**

EXPERIMENT NO: 6

Aim: Implementation of Mutual Exclusion Algorithm.

Theory:

What is mutual exclusion?

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.

Requirements of Mutual exclusion Algorithm:

- **No Deadlock:**
  Two or more sites should not endlessly wait for any message that will never arrive.

- **No Starvation:**
  Every site who wants to execute a 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 are repeatedly executing critical section

- **Fairness:**
  Each site should get a fair chance to execute a critical section. Any request to execute a 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.

Below are the approaches based on message passing to implement mutual exclusion in distributed systems:

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 numbers to order requests for the critical section.
- Each request for the critical section contains a sequence number. This sequence
- number is used to distinguish old and current requests.
- This approach insures Mutual exclusion as the token is unique

Non-token based approach:

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

Example:

Lamport's algorithm, Ricart–Agrawala algorithm

Ricart–Agrawala:

Ricart–Agrawala algorithm is an algorithm for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala. This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm. Like Lamport's Algorithm, it also follows a permission based approach to ensure mutual exclusion.

In this algorithm:

- Two types of messages (**REQUEST** and **REPLY**) are used and communication channels are assumed to follow FIFO order.
- A site sends a **REQUEST** message to all other sites to get their permission to enter a critical section.
- A site sends a **REPLY** message to another site to give its permission to enter the critical section.

- A timestamp is given to each critical section request using Lamport’s logical clock.
- Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section requests is always in the order of their timestamp.

Algorithm:

- **To enter Critical section:**

  When a site S_i wants to enter the critical section, it sends a timestamped **REQUEST** message to all other sites.

  When a site S_j receives a **REQUEST** message from site S_i, It sends a **REPLY** message to site S_i if and only if

- Site S_j is neither requesting nor currently executing the critical section.
- In case Site S_j is requesting, the timestamp of Site S_i‘s request is smaller than its own request.Otherwise the request is deferred by site S_j.
- **To execute the critical section:**

  Site S_i enters the critical section if it has received the **REPLY** message from all other sites.

- **To release the critical section:**

  Upon exiting the site S_i sends a REPLY message to all the deferred requests.

Message Complexity:

Ricart–Agrawala algorithm requires invocation of 2(N – 1) messages per critical section execution. These 2(N – 1) messages involves

- (N – 1) request messages
- (N – 1) reply messages

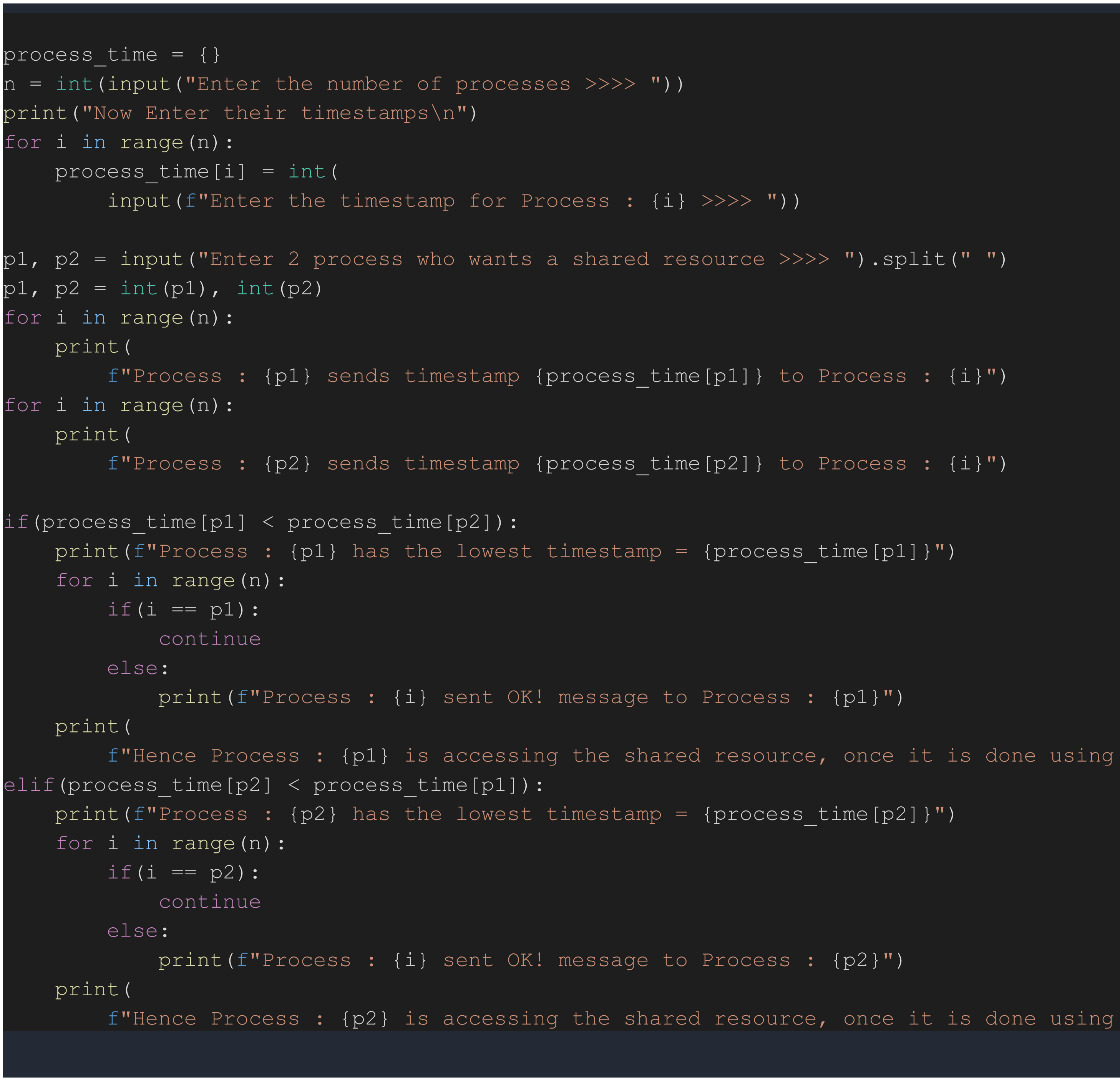
Drawbacks of Ricart–Agrawala algorithm:

- **Unreliable approach:** failure of any one of the nodes in the system can halt the progress of the system. In this situation, the process will starve forever.
  The problem of failure of nodes can be solved by detecting failure after some timeout.

Performance:

- Synchronization delay is equal to maximum message transmission time
- It requires 2(N – 1) messages per Critical section execution

Code:

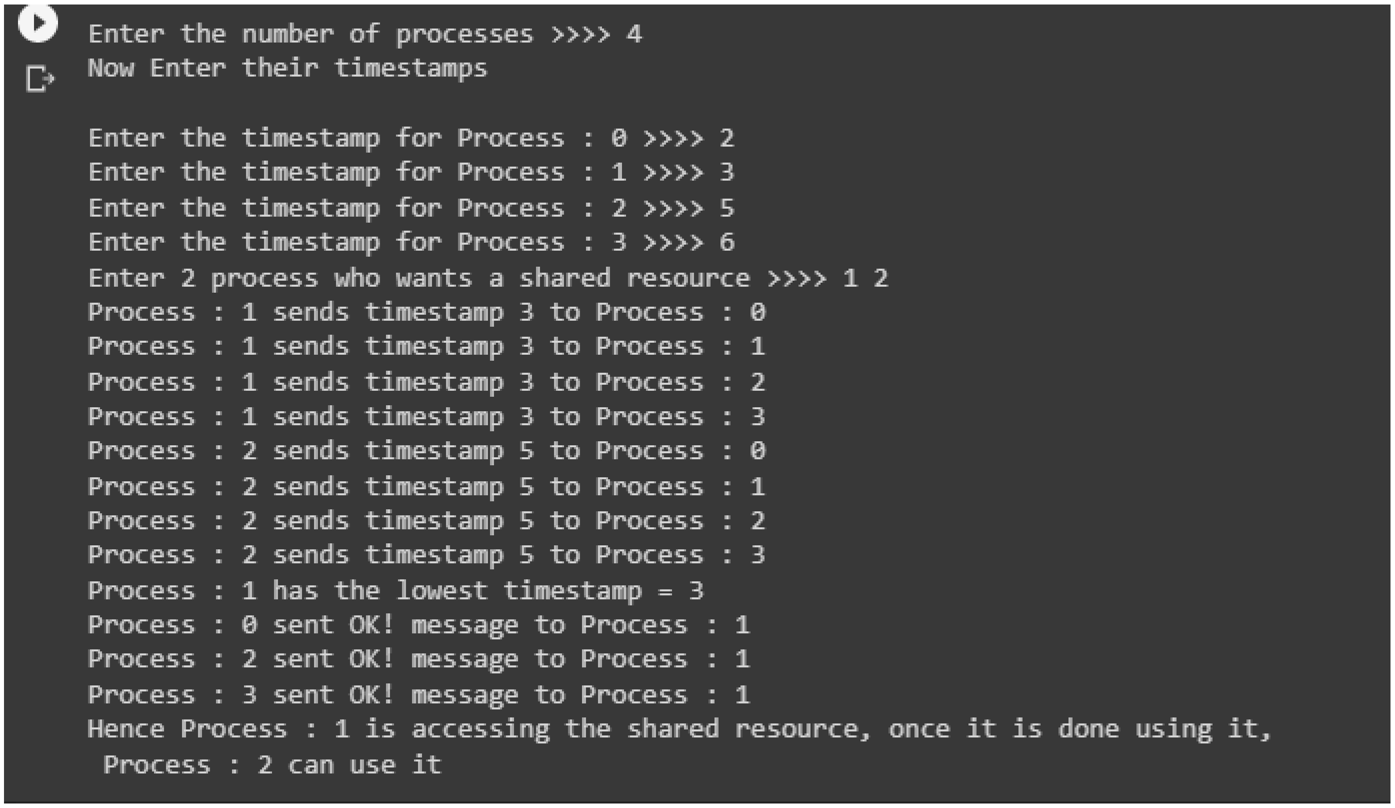


```python
process_time = {}
n = int(input("Enter the number of processes >>>> "))
print("Now Enter their timestamps\n")
for i in range(n):
    process_time[i] = int(
        input(f"Enter the timestamp for Process : {i} >>>> "))

p1, p2 = input("Enter 2 process who wants a shared resource >>>> ").split(" ")
p1, p2 = int(p1), int(p2)
for i in range(n):
    print(
        f"Process : {p1} sends timestamp {process_time[p1]} to Process : {i}")
for i in range(n):
    print(
        f"Process : {p2} sends timestamp {process_time[p2]} to Process : {i}")

if(process_time[p1] < process_time[p2]):
    print(f"Process : {p1} has the lowest timestamp = {process_time[p1]}")
    for i in range(n):
        if(i == p1):
            continue
        else:
            print(f"Process : {i} sent OK! message to Process : {p1}")
    print(
        f"Hence Process : {p1} is accessing the shared resource, once it is done using
elif(process_time[p2] < process_time[p1]):
    print(f"Process : {p2} has the lowest timestamp = {process_time[p2]}")
    for i in range(n):
        if(i == p2):
            continue
        else:
            print(f"Process : {i} sent OK! message to Process : {p2}")
    print(
        f"Hence Process : {p2} is accessing the shared resource, once it is done using
```

Output:



```
Enter the number of processes >>>> 4
Now Enter their timestamps

Enter the timestamp for Process : 0 >>>> 2
Enter the timestamp for Process : 1 >>>> 3
Enter the timestamp for Process : 2 >>>> 5
Enter the timestamp for Process : 3 >>>> 6
Enter 2 process who wants a shared resource >>>> 1 2
Process : 1 sends timestamp 3 to Process : 0
Process : 1 sends timestamp 3 to Process : 1
Process : 1 sends timestamp 3 to Process : 2
Process : 1 sends timestamp 3 to Process : 3
Process : 2 sends timestamp 5 to Process : 0
Process : 2 sends timestamp 5 to Process : 1
Process : 2 sends timestamp 5 to Process : 2
Process : 2 sends timestamp 5 to Process : 3
Process : 1 has the lowest timestamp = 3
Process : 0 sent OK! message to Process : 1
Process : 2 sent OK! message to Process : 1
Process : 3 sent OK! message to Process : 1
Hence Process : 1 is accessing the shared resource, once it is done using it,
 Process : 2 can use it
```

Conclusion:

In this experiment, we learnt about Mutual Exclusion and its type i.e. Token based and Non token based. From that we have implemented the Ricart Agrawala Algorithm. Ricart–Agrawala algorithm is an algorithm for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala. This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm.