Sanjana Asrani
DC

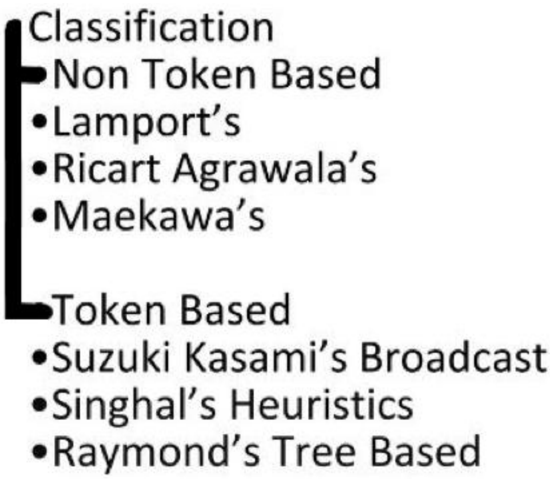
Lab 05
Token Based Algorithm

AIM:

Implement a Token based distributed mutual exclusion. Demonstrate the message overhead complexity by increasing the no of nodes of communication.

THEORY:

Token-based mutual exclusion algorithms rely on the concept of a token, which is a special marker or permission passed among processes to control access to a critical section. These algorithms are designed to ensure that only the process holding the token can enter the critical section, thereby achieving mutual exclusion in a distributed environment. Token-based algorithms typically involve processes communicating directly with each other to request, pass, and release the token as needed.



In Raymond's algorithm, processes are organized in a logical ring, and a token circulates among them. Only the process holding the token can enter the critical section, and after exiting, it passes the token to the next process in the ring.

Singhal's algorithm, proposed by Neeraj Kumar Singhal, is a token-based mutual exclusion algorithm similar to the Suzuki-Kasami algorithm. It also uses logical ring structure for process coordination. However, Singhal's algorithm introduces optimizations to reduce message overhead and improve efficiency compared to Suzuki-Kasami. These optimizations include maintaining additional information about process states and using local knowledge to minimize unnecessary message passing. Singhal's algorithm aims to achieve mutual exclusion with lower communication overhead, making it suitable for large-scale distributed systems.

Suzuki–Kasami algorithm is a token-based algorithm for achieving mutual exclusion in distributed systems.This is modification of Ricart–Agrawala algorithm, a permission based (Non-token based) algorithm which uses REQUEST and REPLY messages to ensure mutual exclusion.
In token-based algorithms, A site is allowed to enter its critical section if it possesses the unique token. Non-token based algorithms uses timestamp to order requests for the critical section where as sequence number is used in token based algorithms.
Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.

Data structure and Notations:

An array of integers RN[1…N]
A site Si keeps RNi[1…N], where RNi[j] is the largest sequence number received so far through REQUEST message from site Si.
An array of integer LN[1…N]
This array is used by the token.LN[J] is the sequence number of the request that is recently executed by site Sj.
A queue Q
This data structure is used by the token to keep record of ID of sites waiting for the token
Algorithm:

To enter Critical section:
When a site Si wants to enter the critical section and it does not have the token then it increments its sequence number RNi[i] and sends a request message REQUEST(i, sn) to all other sites in order to request the token.
Here sn is update value of RNi[i]
When a site Sj receives the request message REQUEST(i, sn) from site Si, it sets RNj[i] to maximum of RNj[i] and sn i.e RNj[i] = max(RNj[i], sn).

After updating RNj[i], Site Sj sends the token to site Si if it has token and RNj[i] = LN[i] + 1

To execute the critical section:

Site Si executes the critical section if it has acquired the token.

To release the critical section:

After finishing the execution Site Si exits the critical section and does following:

sets LN[i] = RNi[i] to indicate that its critical section request RNi[i] has been executed

For every site Sj, whose ID is not present in the token queue Q, it appends its ID to Q if RNi[j] = LN[j] + 1 to indicate that site Sj has an outstanding request.

After above updation, if the Queue Q is non-empty, it pops a site ID from the Q and sends the token to site indicated by popped ID.

If the queue Q is empty, it keeps the token

Message Complexity:

The algorithm requires 0 message invocation if the site already holds the idle token at the time of critical section request or maximum of N message per critical section execution. This N messages involves (N – 1) request messages, 1 reply message

Drawbacks of Suzuki–Kasami Algorithm:

Non-symmetric Algorithm*:* A site retains the token even if it does not have requested for critical section. According to definition of symmetric algorithm: "No site possesses the right to access its critical section when it has not been requested."

Performance:

Synchronization delay is 0 and no message is needed if the site holds the idle token at the time of its request.

In case site does not holds the idle token, the maximum synchronization delay is equal to maximum message transmission time and a maximum of N message is required per critical section invocation.

OUTPUT:

Start all the nodes:

```
PS D:\Engineering_codes\Div-B_01_Sanjana A
srani\sem 8\DC\exp 5\exp 5>   python node1
.py
Server started on port 50051
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue re
quest (or 'exit' to quit):
```

```
PS D:\Engineering_codes\Div-B_01_Sanjana A
srani\sem 8\DC\exp 5\exp 5> python node2.p
y
Server started on port 50052
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue re
quest (or 'exit' to quit):
```

```
PS D:\Engineering_codes\Div-B_01_Sanjana
 Asrani\sem 8\DC\exp 5\exp 5> python nod
e3.py
Server started on port 50053
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
request (or 'exit' to quit):
```

Node1 requests:

Since no one is in the request queue yet, node1 gets entry in C.S. directly.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                python

PS D:\Engineering_codes\Div-B_01_Sanjana
 Asrani\sem 8\DC\exp 5\exp 5> python nod
e1.py
Server started on port 50051
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
request (or 'exit' to quit): 1
0
True
Node 1 Entered Critical Section
----------------------------------------
----
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
request (or 'exit' to quit):
```

```
PS D:\Engineering_codes\Div-B_01_Sanjan
a Asrani\sem 8\DC\exp 5\exp 5> python n
ode2.py
Server started on port 50052
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
 request (or 'exit' to quit):
```

```
PS D:\Engineering_codes\Div-B_01_Sanja
na Asrani\sem 8\DC\exp 5\exp 5> python
 node3.py
Server started on port 50053
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queu
e request (or 'exit' to quit):
```

Now suppose node2 requests while node1 is in C.S.

```
PS D:\Engineering_codes\Div-B_01_Sanjana
 Asrani\sem 8\DC\exp 5\exp 5> python nod
e1.py
Server started on port 50051
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
request (or 'exit' to quit): 1
0
True
Node 1 Entered Critical Section
----------------------------------------
----
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
request (or 'exit' to quit):
```

```
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
 request (or 'exit' to quit): 1
0
False
Wait until you recieve token
Request:data: "2"

Get Queue:['2']
Request:data: "2"

Request:data: "2"

1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
 request (or 'exit' to quit):
```

```
PS D:\Engineering_codes\Div-B_01_Sanja
na Asrani\sem 8\DC\exp 5\exp 5> python
 node3.py
Server started on port 50053
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queu
e request (or 'exit' to quit):
```

So now the request queue has node2

Now node3 requests:

```
PS D:\Engineering_codes\Div-B_01_Sanjana
 Asrani\sem 8\DC\exp 5\exp 5> python nod
e1.py
Server started on port 50051
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
request (or 'exit' to quit): 1
0
True
Node 1 Entered Critical Section
----------------------------------------
----
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
request (or 'exit' to quit):
```

```
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
 request (or 'exit' to quit): 1
0
False
Wait until you recieve token
Request:data: "2"

Get Queue:['2']
Request:data: "2"

Request:data: "2"

1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
 request (or 'exit' to quit):
```

```
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queu
e request (or 'exit' to quit): 1
0
False
Wait until you recieve token
Request:data: "3"

Get Queue:['2', '3']
Request:data: "2"
data: "3"

Request:data: "2"
data: "3"

1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queu
e request (or 'exit' to quit):
```

So request queue has node2, node3

As soon as node1 now finishes execution in C.S. and exits,
Node2 is allowed to get in C.S.

```
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
request (or 'exit' to quit): 2
Release Critical Section Initialized
Critical Section Released by Node 1
```

```
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queue
 request (or 'exit' to quit): Node 2 En
tered Critical Section
----------------------------------------
-----
```

```
1. Request for Critical Section
2. Release the Critical Section
3. Check Queue
4. Check Sequence number
5. Change Token Status
6. Type 'Exit' to terminate
Enter 1 to add information in the queu
e request (or 'exit' to quit): Node 3
Entered Critical Section
----------------------------------------
------
2
Release Critical Section Initialized
Release Critical Section by Node 3
No other Node requesting for Critical
Section
```

Only when node2 releases, node3 gets to enter:

```
--
2
Release Critical Section Initialized
Release Critical Section by Node 2
```

CONCLUSION:

In conclusion, the Suzuki-Kasami algorithm has been successfully implemented using gRPC for inter-process communication. Despite its effectiveness, the algorithm exhibits high message complexity, typically O(n^2) messages for n processes, which can impact scalability.