Date: 13/Febuary/2023

M T W T F S S

NAME	:	JAWAD AHMED
ROLL NO	:	20P-0165
Section	:	BCS-6A

Subject: Parallel Distributed Computing

Assignment # 05

X ———————— X

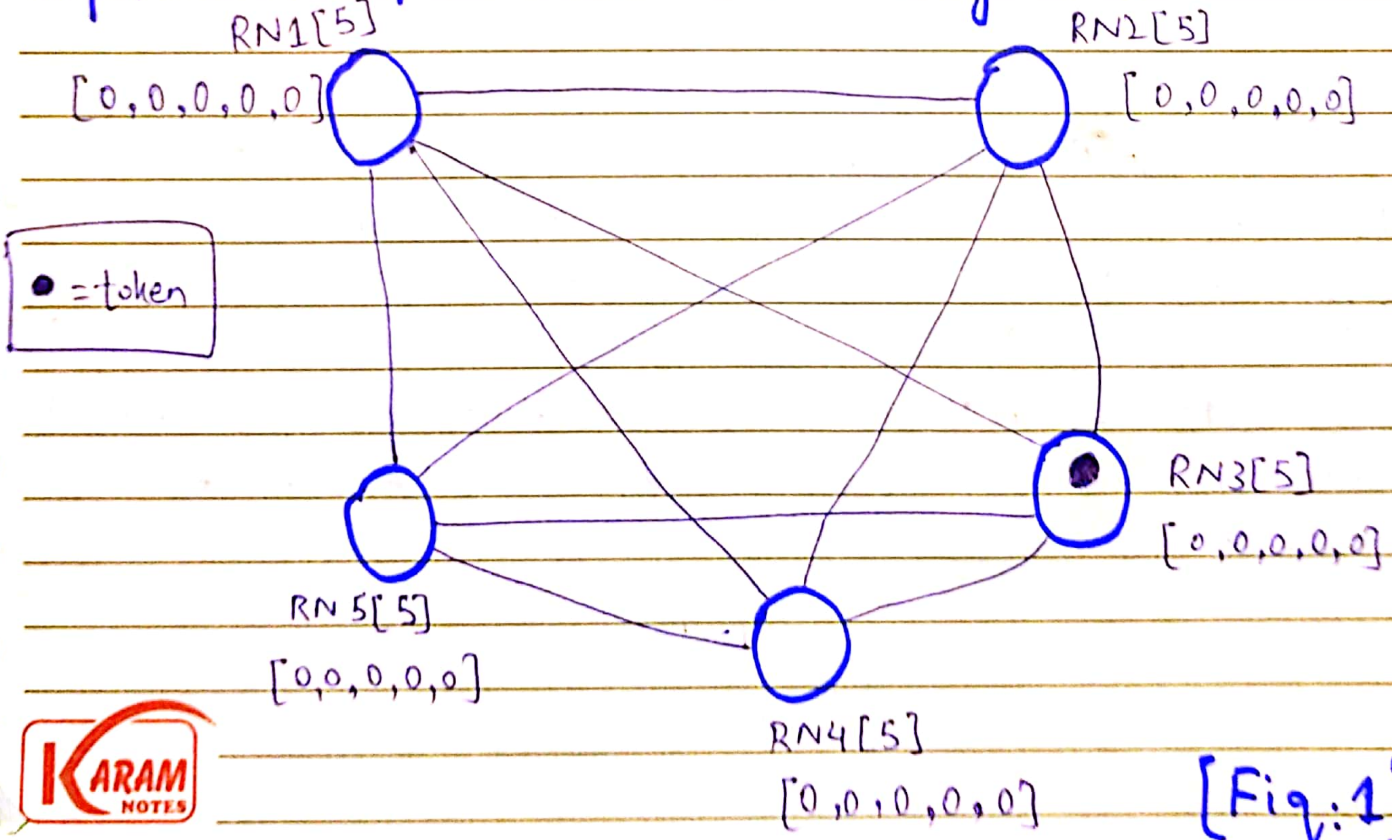
Date:

Q:- Explain Suzuki-Kasami Algorithm for mutual Exclusion in Distributed System. Write Code for it?

Ans: Suzuki-Kasami Algorithm is a token-based Algorithm for achieving mutual exclusion in distributed systems.

In token-based algorithms, A site is allowed to enter its critical section if it possesses the unique token. Sequence Number is used to order requests for the critical section. Sequence number is used to distinguish old and current requests. Suzuki Kasami algorithm works in a Completely connected network of processes.

Explanation of Suzuki-Kasami Algorithm



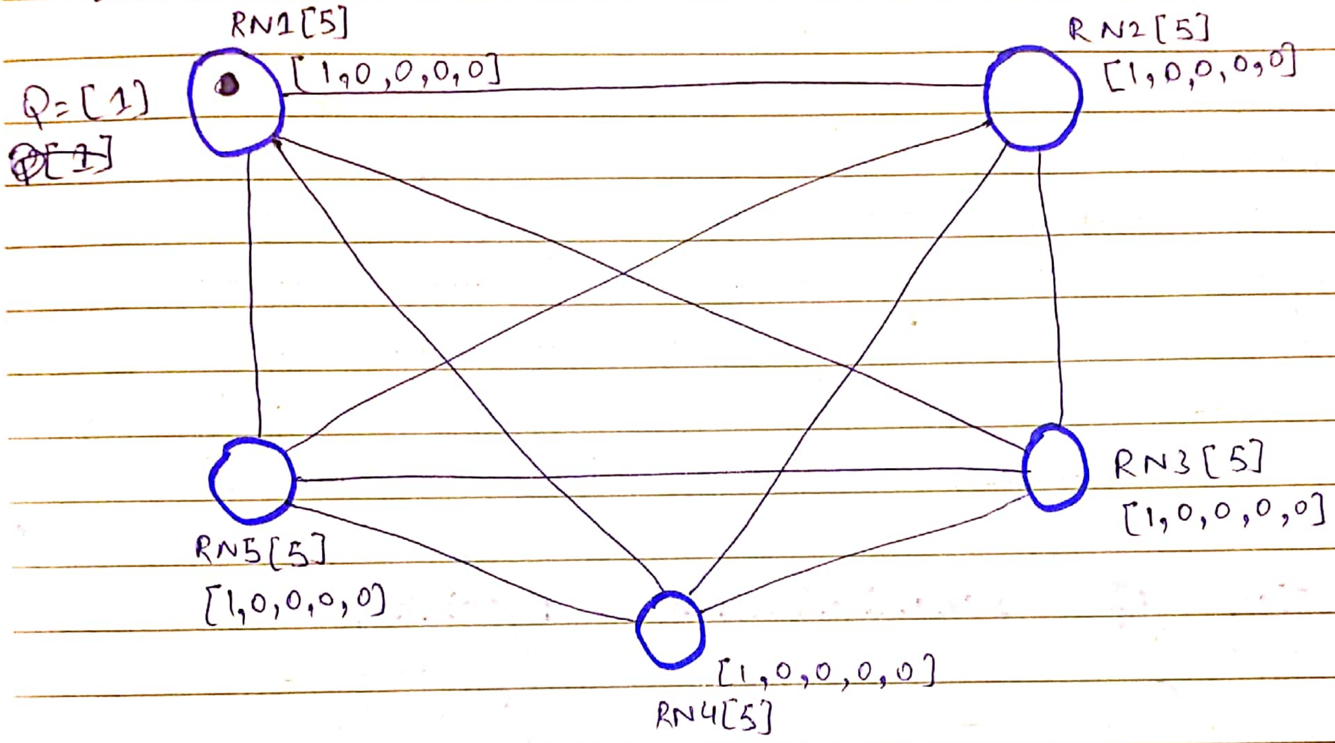
[Fig:1]

Scanned with CamScanner

Date:

① RN1[5] "[5]" indicates "Number of Nodes or sites it is linked to." In this example it has five interconnected nodes.

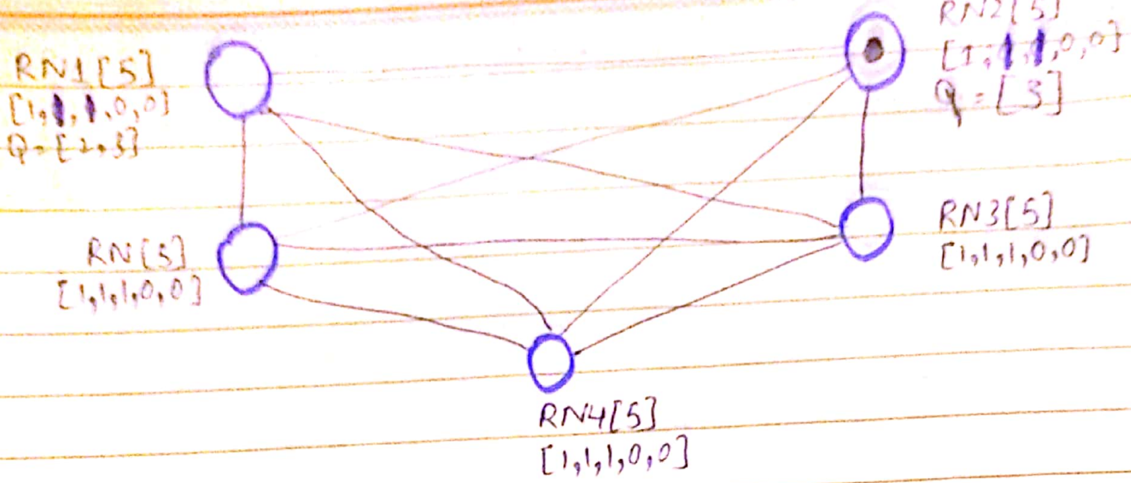
② Second we have an array filled with zeros that mean No one has made a request for token Now.



③ The process ① has requested to enter the critical the message will be broadcasted to every process and the request will be get stored in Queue.

④ The token is held by RN3 shown in fig 1. The RN3 not in it's critical section so token will be given to the RN1.

Date:



5 Now the process 2 & 3 requested for a token so both processes will be placed in queue and the request message will be broadcasted as shown in the array (sequence number updated).

6 RN1 is done with the critical section so token will given to RN2 and message will be broadcasted.

7 When RN2 done with the critical section then token will be given to RN3 and message will be broadcasted.

This process will continue until no deadlock happens.
This is a broadcast based algorithm. (Token based)

Date:

Python Code For Suzuki-Kasami Algorithm

```
class Process:
    def __init__(self, id, n, request_queue):
        self.id = id
        self.n = n
        self.request_queue = request_queue
        self.token = False
        self.request_sent = [False] * n
        self.granted = [False] * n

    def run(self):
        while True:
            if self.token:
                print(f"Process {self.id} has the token")
                for i in range(self.n):
                    if not self.granted[i]:
                        print(f"Process {self.id} send request to process {i}.")
                        self.request_sent[i] = True
                        self.request_queue[i].append(self.id)
                self.token = False
                print(f"Process {self.id} releases the token")
            else:
                for i in range(self.n):
                    if self.request_sent[i] and not self.granted[i]:
                        if self.request_queue[i][0] = self.id:
                            print(f"Process{i} granted access to process {self.id}")
                            self.granted[i] = True
                            self.request_queue[i].pop(0)
                            self.token = True
                            break
```

Date:

```
def   test_suzuki_kasami( ):
      n= 5
      request_queve = [[] for i in range(n)]
      processes  = [Process(i, n, request_queve) for in in range(5)]

      for  process  in  processes:
           process.run()
```

Q2:- Explain Maekawa's Algorithm? Write a code for it.

Ans: Maekawa's algorithm is quorum based approach to ensure mutual exclusion in distributed systems. In maekawa's algorithm one process does not request permission from every other ~~site~~ ~~but~~ processes but from a subset of processes which is called quorum.

Types of Messages

① **REQUEST:** A site send a REQUEST message to all other ~~site~~ processes in its request set or quorum to get their permission to enter critical section.

② **REPLY:** A site send a REPLY message to requesting process to give its permission to enter the critical section

Date:............

③ RELEASE: A site sends a RELEASE message to all other ~~site~~ process in its request set or quorum upon exiting the critical section.

Assumption And RULES

→ Every process should belong to some group.

→ If any process wants to ~~acc~~ access critical section then it will send request to its group members and recieve reply from group members.

Rules For Process

- The Intersection of two group != NULL.
- The process should be a part of any group.
- Single process cannot be considered as a group.

quorum 1: 1 2 3 4
quorum 2: 6 7 3

Python Code For Maekawa's Algorithm

```
class Process:
    def __init__(self, id, num_processes):
        self.id = id
        self.num_processes = num_processes
        self.timestamp = 0
        self.queue = [ ]
        self.in_cs = False
        self.replies_expected = 0
```

Date:............

```
def  enter_cs(self):
     self.timestamp ~~1~~ += 1
     self.in_cs = True
     self.replies_executed = self.num-processes - 1
     request = (self.id, self.timestamp)
     for i in range(self.num_processes):
          if i != self.id:
               self.send_request(i, request)

def  recieve_request(self, request):
     sender_id, sender_timestamp = request
     if (not self.in_cs) or (self.in_cs and sender_timestamp < self.time_stamp)
          self.queue.append(request)
          self.queue.sort(key= lambda x: x[1])
          self.send_reply(sender_id)
     else:
          self.queue.append(request)

def  send_request(self, reciever_id, request):
          self.replies_expected -= 1
          if (self.replies_expected == 0):
               self.enter_critical_section()

def  send_request(self, reciever_id, request):
          pass

def  send_reply(self, reciever_id):
          pass
```

Scanned with CamScanner

Date:

```
def     enter_critical_section(self):
            pass

def     test_maekawa_algorithm():
        num_processes = 5
        processes = [Process(i, num_processes) for i in range(num_processes)]

        #Test1 : Each Process Enters & exits critical section 1 by 1
        for p in processes:
                p.enter_cs()
                assert p.in_cs = True
                p.in_cs = False

        #Test 2: All processes try to enter critical section at same Time
        for p in processes:
                p.enter_cs()
        for p in processes:
            assert p.in_cs = True
        # Exit Critical Section
        for p in processes:
            p.in_cs = False
```

KARAM NOTES