GOA COLLEGE OF ENGINEERING

"Bhausaheb Bandodkar Technical Education Complex"

Experiment No: 7

Distributed Deadlocks

Aim: To implement Distributed deadlocks.

Theory:

Distributed deadlocks can occur when distributed transactions or concurrency control are utilized in distributed systems. It may be identified via a distributed technique like edge chasing or by creating a global wait-for graph (WFG) from local wait-for graphs at a deadlock detector. Phantom deadlocks are identified in a distributed system but do not exist due to internal system delays.

In a distributed system, deadlock cannot be prevented nor avoided because the system is too vast. As a result, only deadlock detection is possible. The following are required for distributed system deadlock detection techniques:

1. Progress

- The method may detect all the deadlocks in the system.

2. Safety

- The approach must be capable of detecting all system deadlocks.
- Approaches to detect deadlock in the distributed system

Various approaches to detect the deadlock in the distributed system are as follows:

1. Centralized Approach

Only one resource is responsible for detecting deadlock in the centralized method, and it is simple and easy to use. Still, the disadvantages include excessive workload on a single node and single-point failure (i.e., the entire system is dependent on one node, and if that node fails, the entire system crashes), making the system less reliable.

2. Hierarchical Approach

In a distributed system, it is the integration of both centralized and distributed approaches to deadlock detection. In this strategy, a single node handles a set of selected nodes or clusters of nodes that are in charge of deadlock detection.

3. Distributed Approach

In the distributed technique, various nodes work to detect deadlocks. There is no single point of failure as the workload is equally spread among all nodes. It also helps to increase the speed of deadlock detection.

Deadlock Handling Strategies

Various deadlock handling strategies in the distributed system are as follows:

1. There are mainly three approaches to handling deadlocks: deadlock prevention, deadlock avoidance, and deadlock detection.
2. Handling deadlock becomes more complex in distributed systems since no site has complete knowledge of the system's present state and every inter-site communication entails a limited and unpredictable latency.
3. The operating system uses the deadlock Avoidance method to determine whether the system is in a safe or unsafe state. The process must inform the operating system of the maximum number of resources, and a process may request to complete its execution.
4. Deadlocks prevention are commonly accomplished by implementing a process to acquire all of the essential resources at the same time before starting execution or by pre-empting a process that already has the resource.
5. In distributed systems, this method is highly inefficient and impractical.

Program:

```
import time
class Process:
   def __init__(self,name='A',holder=None):
      self.name=name
      self.cs=False
      if holder is  None:
         self.holder=self
      else:
         self.holder=holder
      self.reqQ=[]
      self.asked=False


   def assignToken(self):
      if self.holder!=self and self.cs==False and len(self.reqQ)>0:
         self.holder=self.reqQ.pop(0)
         self.asked=False
      if self.holder==self:
         self.cs=True
      #else:
         #self.sendToken(self.holder)
```

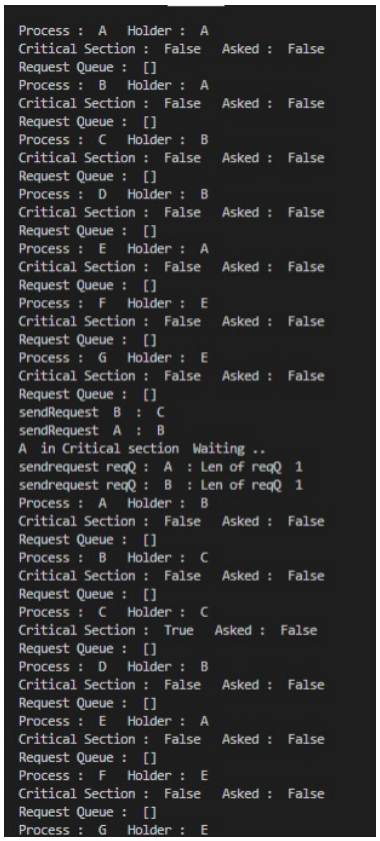
```
    def sendRequest(self,neighbor):
        print("sendRequest ",self.name," : ",neighbor.name)
        if self.cs==False and self!=self.holder :
            self.reqQ.append(neighbor)
            self.holder.sendRequest(self)
        elif self==self.holder:
            self.reqQ.append(neighbor)
            self.releaseResource()
        print("sendrequest reqQ : ",self.name," : Len of reqQ ",len(self.reqQ))
        self.holder=self.reqQ.pop(0)


    def releaseResource(self):
        if self.cs==True:
            print(self.name, " in Critical section  Waiting ..")
            time.sleep(2)
            self.cs=False

def makeRequest(self):
        if self!=self.holder and self.cs==False and len(self.reqQ)==0:
            self.reqQ.append(self)
            self.asked=True
            self.holder.sendRequest(self)
            self.assignToken()



    def setInfo(self,name,holder,cs):
        self.holder=holder
        self.name=name
        self.holder=holder
        self.cs=cs


    def show(self):
        print("Process : ",self.name,"  Holder : ",self.holder.name)
```

```
        print("Critical Section : ",self.cs,"  Asked : ",self.asked)
        print("Request Queue : ",self.reqQ)

a=Process('A')
b=Process('B',a)
c=Process('C',b)
d=Process('D',b)
e=Process('E',a)
f=Process('F',e)
g=Process('G',e)


processQ=dict()


processQ['A']=a
processQ['B']=b
processQ['C']=c
processQ['D']=d
processQ['E']=e
processQ['F']=f
processQ['G']=g


root=processQ['A']


for x in processQ:
    processQ[x].show()


processQ['A'].cs=True
processQ['C'].makeRequest()


for x in processQ:
    processQ[x].show()
```

GOA COLLEGE OF ENGINEERING

"Bhausaheb Bandodkar Technical Education Complex"

Output:



Conclusion: Distributed deadlocks was successfully implemented ad executed.