Project #2 Report

CSC 8980: Distributed Systems

(Group #3 -** Pooja Baba, Ishu Goyal, Dinesh Kumar Pullepally, Samvidha Reddy Mannem**)

Problem Statement

Implement Maekawa's mutual exclusion algorithm to **coordinate mutual exclusive access to a single resource**. Given a distributed system with n nodes, you must **first use the result of Project #1** to **compute the sets** used to implement Maekawa's algorithm. Then develop a simulation in which individual nodes request mutual exclusive access to a shared resource.

Simulation Implementation files

The implementation is done using **python** language. Following are the files used in the system to generate the sets and to implement Maekawa's mutual exclusion algorithm -

1. config.py
   This file is created in project 1 after all the final sets have been created and the user exits the system (by selecting option#3 in the list). The file includes the following parameters, which are used throughout the mutual exclusion algorithm implementation -

   - NUM_NODE: indicates the number of nodes in the system
   - INIT_PORT: port number of the first node
   - NODE_PORT = consecutive init ports for the other nodes in the system
   - RECV_BUFFER: socket buffer size when receiving messages
   - REQUEST_SETS = list of dictionaries for request sets. This list will be generated by project #1 and stored in the config file

2. enum.py
   This file defines some enum types for the project. This file consists of the following –

   - STATE: this defines the state the node holds at one instant of the time. This has further values:-
     - INIT (0)
     - REQUEST (1)
     - HELD (2)
     - RELEASE (3)
   - MSG_TYPE: this defines the type of message that is being sent or received by a node. This has further values:-
     - REQUEST (0)
     - GRANT (1)
     - RELEASE (2)
     - FAIL (3)
     - INQUIRE (4)
     - YEILD (5)

3. logging.py
   This file consists of functions that each node will print on the console as and when the algorithm implementation proceeds. The function prints the timestamp, thread_id (node_id), src (node from which the message is received) and the type of the message. The output details are in the following sections.

4. maekawa.py
   This file when executed, creates the system setup. It requires the following parameters which are assigned to every node –

   - cs_int = 4
   - next_req = 7

- option (if any)

5. message.py
   This file consists of functionalities that require creating messages that each node sends to each other for communication. Message encoding and decoding are a part of the functionalities of this class. This requires the source node id, destination node id, data, and the message type being communicated in the system.

6. mutex.py
   This is the main program entrance for the execution of the mutual exclusion system. Following arguments are required for each thread (node) that will be created in this file –

   - cs_int = Time each node will take to execute in the critical section. (In our code it is 4ms)
   - next_req = Time interval between two nodes to generate request messages. (In our code it is 7)
   - total_execution_time = Total time for the execution of the entire system. (In our code it is 15)

7. node.py
   This file has code that needs to be executed for each node that will be created. This involves the following functions –

   - Communicating with other nodes via message passing
   - Competing for critical section
   - Creating the voting set using the REQUEST_SETS fetched from the *config.py* file
   - Building connections between the nodes
   - Entering the critical section
   - Exiting the critical section
   - Creating request critical section messages
   - Handling the request received by the node, i.e, what needs to be done if the node receives a REQUEST message, RELEASE message, etc.

8. utils.py
   This file has basic utility functions such as creating a server socket, creating a client socket and converting DateTime to string value to print the timestamp in human-readable format on the screen.

Interaction between the files

In this section, we will talk about the interaction between the files for implementing mutual exclusive access of the critical section.

1. As provided in the files' description in the above section, *mutex.py* serves as the entry point for the mutual exclusion simulation. It creates an entry point to the Maekawas.py file's **MaekawaMutex** class.
2. Using the run method from **MaekawaMutex** class (*maekawa.py* file), it creates connection between the nodes. Here it uses the run method from **Node** class (*node.py* file). Here the number of nodes value for configuring the nodes data is fetched from the *config.py* file.
3. In *node.py* file there are multiple classes created as follows –
   a. **ServerThread**: This class implements receiving and processing messages as server. The main methods in this class are the update and process_message.
      i. Update method: This method as the name suggests, receives the message from the client and processes the message using the **Message** class and works on the sending part. Also receives the message and updates it accordingly.
      ii. Process_message: Upon receiving the message from the server socket, this method checks for the type of the message using the MSG_TYPE enum from the *enum.py* file and implements that method accordingly.
   b. **ClientThread**: This class implements sending message as a client. This is in charge of sending request message to the other nodes. This class makes use of the multicast method to multicast the message details to a group of destination node ids.
   c. **DelayThread**: This class implements message passing delay. This is mainly used for debug purpose
   d. **Node**: This class encapsulates all the functionalities of a node as described.
      i. Creates its voting set using the *config.py* file.

ii. Checks for the number of votes received
iii. Checks for the request_queue for further processing of the next node to enter the critical section
iv. Uses check method to determine the state of the node using the STATE enum from the *enum.py* file.
v. Takes care of the lamport timestamp while entering critical section, exiting critical section and sending requests to other nodes.

Code Implementation for Maekawa mutual exclusion access

1. Project #1 sets generation code output –
   The screenshot below shows the sets generated after executing the Project #1 code. This output is the input for Project #2. The final sets generated are based on **7** number of nodes

```
***** Executing Project 1 *****

Enter number of nodes: 4

4 cannot be expressed as k*k - k + 1

Finding new temporary N...
Temporary N found is: 7
Therefore we will create sets of size 3 from 7 different nodes.

************ Creating Sets ************

Received N and K as: 7 and 3
Starting to create sets...


Set Creation with new value of 'N' complete...
******* Degenerate Sets Creation *******

Starting Degenerate Sets Creation...

Set 1  : ['1', '2', '3']
Set 2  : ['2', '4']
Set 3  : ['3', '4']

*******Final Degenerate Set Creation Completed*******
*******Set Creation Completed*******



*******Set Menu*******
1. To Add Node/Nodes
2. To Remove Node/Nodes
3. Exit
Enter option: 1
Enter number of nodes to be added: 3
7 can be expressed as k*k - k + 1


************ Creating Sets ************

Received N and K as: 7 and 3
Starting to create sets...

Set 1  : ['1', '2', '3']
Set 2  : ['1', '4', '5']
Set 3  : ['1', '6', '7']
Set 4  : ['2', '4', '6']
Set 5  : ['2', '5', '7']
Set 6  : ['3', '4', '7']
Set 7  : ['3', '5', '6']


*******Set Creation Completed*******

*******Set Menu*******
1. To Add Node/Nodes
2. To Remove Node/Nodes
3. Exit
Enter option: 3
Config.Py File Created
```

2. Project #2 Maekawa mutual exclusion algorithm output

The screenshot below shows the mutual exclusion algorithm output for the sets generated by Project #1. The output can be explained as –

Example:

Output 1 – Communication between the nodes using the MESSAGE class

Time	Node ID	Sender ID	Message Type
14:44:32:793	1	0	REQUEST

Interpretation - Node with ID **0** sends a message **REQUEST** to node with ID **1** at timestamp **14:44:32:793**

Output 2 – Details of the node entering the critical section

Time	Node ID	Node List
14:44:32:795	0	[1, 2, 3]

Interpretation – Node with ID **0** enters the critical section at system time **14:44:32:795** after receiving permission from the nodes present in the node list **[1, 2, 3]**

```
***** Switching to Project 2 *****
14:44:32:793 1 0 REQUEST
14:44:32:793 3 0 REQUEST
14:44:32:793 0 1 GRANT
14:44:32:793 2 0 REQUEST
14:44:32:793 0 3 GRANT
14:44:32:793 0 2 GRANT
14:44:32:794 1 1 REQUEST
14:44:32:794 5 1 REQUEST
14:44:32:794 6 2 REQUEST
14:44:32:794 4 1 REQUEST
14:44:32:794 2 3 REQUEST
14:44:32:794 1 1 FAIL
14:44:32:794 4 3 REQUEST
14:44:32:795 1 2 REQUEST
14:44:32:795 3 2 FAIL
14:44:32:795 5 4 REQUEST
14:44:32:795 3 4 FAIL
14:44:32:795 2 4 REQUEST
14:44:32:795 1 5 GRANT
14:44:32:795 6 3 REQUEST
14:44:32:795 4 5 FAIL
14:44:32:795 3 5 REQUEST
14:44:32:795 2 6 GRANT
14:44:32:795 4 2 FAIL
14:44:32:795 3 6 FAIL
14:44:32:795 2 1 FAIL
14:44:32:795 0 [1, 2, 3]
14:44:32:796 1 4 GRANT
14:44:32:796 4 5 REQUEST
14:44:32:796 5 3 FAIL
14:44:32:796 3 6 REQUEST
14:44:32:796 5 6 REQUEST
14:44:32:796 6 6 REQUEST
14:44:32:796 5 4 FAIL
14:44:32:796 6 3 FAIL
14:44:32:796 6 5 FAIL
14:44:32:796 6 6 FAIL
14:44:32:801 1 0 RELEASE
14:44:32:801 1 1 GRANT
14:44:32:801 3 0 RELEASE
14:44:32:801 2 0 RELEASE
14:44:32:801 6 3 GRANT
14:44:32:801 3 2 GRANT
14:44:32:802 1 [1, 4, 5]
14:44:32:807 4 1 RELEASE
14:44:32:807 5 1 RELEASE
14:44:32:807 1 1 RELEASE
14:44:32:807 3 4 GRANT
14:44:32:807 2 1 GRANT
14:44:32:807 4 5 GRANT
14:44:32:808 2 0 REQUEST
14:44:32:808 1 0 REQUEST
14:44:32:809 0 2 FAIL
14:44:32:809 0 1 FAIL
14:44:32:809 3 0 REQUEST
14:44:32:809 6 3 INQUIRE
14:44:32:809 3 6 YIELD
14:44:32:809 0 3 GRANT
14:44:32:815 4 1 REQUEST
14:44:32:815 5 1 REQUEST
14:44:32:815 1 1 REQUEST
14:44:32:815 1 1 FAIL
14:44:32:815 1 4 FAIL
14:44:32:815 1 5 FAIL
(base) ishug@Ishus-MacBook-Air final %
```