Spring 2015 ***CS 425 - Distributed Systems*** Shubham Agarwal
Date: 30th April, 2015 ***MP3 Report*** sagarwl4

COMMAND TO RUN CODE:

python maekawa.py <cs_int> <next_req> <tot_exec_time> <option>

Here cs_int = some integer
next_req = some integer
tot_exec_time = some integer
option = 1 or 0

IMPLEMENTATION DETAILS:

Has 9 different threads/processes which have all their resources to themselves, i.e. none of them are shared.

The design has been laid out like a grid like this:

1	2	3
4	5	6
7	8	9

The voting set for any process it the elements in its row and column.

Process No.	Voting set				
1	1	2	3	4	7
2	1	2	3	5	8
3	1	2	3	6	9
4	1	4	5	6	7
5	2	4	5	6	8
6	3	4	5	6	9
7	1	4	7	8	9
8	2	5	7	8	9
9	3	6	7	8	9

HANDLING DEADLOCKS:

Each process has a listening section which queues the incoming packet immediately. This is to prevent packets coming one after another to be dropped. It spawns one thread in the beginning which continuously waits until the first packet arrives. After the first packet arrives, it waits for 100 milliseconds. I assume that packets destined for this particular port would be delivered within these 100 millisecond. The thread then runs through the list and picks up the packet with the lowest timestamp. Every packet has a timestamp that was generated at the time of sending the packets. As there is only one system, the time can be directly compared. Now this packet is sent a response and popped from list. Thus, in most cases, the process

which tried to send a message before everyone else gets a reply before everyone else. Thus, we prevent any deadlocks from happening in the first place.
These is also another thread called deadlock detection thread which keeps running in the background and keeps checking if there is more than one thread in the HELD state. In case there is, it prints out that a deadlock has been detected and proceeds to clear all the lists and resets all the counters to 0. This way, things start from the beginning. Although this deadlock is not easy to reach and the deadlock thread rarely has to take actions.

SAMPLE OUTPUT AFTER EXECUTING THE PROGRAM

counter6 = 5
Process6CS -> Time:%s, Thread_identifier:node6, node-list: 3 4 5 6 9 2015-04-29 16:19:28.887880
counter5 = 5
Process5CS -> Time:%s, Thread_identifier:node5, node-list: 2 4 5 6 8 2015-04-29 16:19:33.398768
counter7 = 5
Process7CS -> Time:%s, Thread_identifier:node7, node-list: 1 4 7 8 9 2015-04-29 16:19:37.909674
counter9 = 5
Process9CS -> Time:%s, Thread_identifier:node9, node-list: 3 6 7 8 9 2015-04-29 16:19:42.421828
counter3 = 5
Process3CS -> Time:%s, Thre
ad_identifier:node3, node-list: 1 2 3 6 9 2015-04-29 16:19:46.932255
counter2 = 5
Process2CS -> Time:%s, Thread_identifier:node2, node-list: 1 2 3 5 8 2015-04-29 16:19:51.443203
counter8 = 5
Process8CS -> Time:%s, Thread_identifier:node8, node-list: 2 5 7 8 9 2015-04-29 16:19:55.955267
counter1 = 5
Process1CS -> Time:%s, Thread_identifier:node1, node-list: 1 2 3 4 7 2015-04-29 16:20:00.468109
counter4 = 5
Process4CS -> Time:%s, Thread_identifier:node4, node-list: 1 4 5 6 7 2015-04-29 16:20:04.982093