CSE 535 Assignment 2

Saish Sali (SBU ID: 111492587)

Problem 1: Lamport's distributed mutex exclusion algorithm implementation

- The algorithm is implemented following five rules described in Lamport's CACM 1978 time-clocks paper.
- Assumption: "Any" is assumed to be any one Tm:Pi request resource message from request queue.
- Each process requests a resource, enters critical section and then releases resource following five rules mentioned in Lamport's logical time clock paper.
- Each process interleaves request and release commands.
- The algorithm has been implemented in **lamport.da** file.
- Compile program:

  python -m da.compiler lamport.da
- Execute program:

  python -m da lamport.da <num_procs> <num_requests>

Problem 2: Safety and liveness violation

- **Liveness violation:**
  - Let us consider 2 processes in a distributed system with process ids P1 and P2 respectively. Assumption: any means any one message in the request queue.
  - To request the resource, process P sends the message x:P request resource to every other process and puts that message into its request queue, where x is the timestamp of the message.
  - Let us consider process P1 sends resource request command twice and P2 sends request command once such that queue state is as follows:

1:P1	2:P2	3:P1

Fig 1(a): Request queue for process P1

1:P1	2:P2	3:P1

Fig 1(b): Request queue for process P2

  - Let us assume P1 has been granted the resource. To release the resource, P1 removes **1:P1** request message from its queue and sends release resource message to every other process.

2:P2	3:P1

Fig 1(c): Request queue for process P1 after removing 1:P1

- When process P2 receives the P1 release resource message, it removes **3:P1** request message from its queue.

1:P1	2:P2

Fig 1(d): Request queue for process P2 after removing 3:P1

- Since process P1 has 2:P2 as the head of the queue and P2 has 1:P1 as the head of the queue, there will be a deadlock (thus liveness violation) because there is no request message of respective processes in the request queue which is ordered before any other request in its queue (Rule 5).

- **Safety violation:**
  - Let us consider 2 processes in a distributed system with process ids P1 and P2 respectively. Assumption: any means any one message in the request queue.
  - To request the resource, process P sends the message x:P request resource to every other process and puts that message into its request queue, where x is the timestamp of the message.
  - Let us consider process P1 sends resource request command twice and P2 sends request command once such that queue state is as follows:

1:P1	2:P2	3:P1

Fig 2(a): Request queue for process P1

1:P1	2:P2	3:P1

Fig 2(b): Request queue for process P2

- Let us assume P1 has been granted the resource. To release the resource, P1 removes **3:P1** request message from its queue and sends release resource message to every other process.

1:P1	2:P2

Fig 2(c): Request queue for process P1 after removing 3:P1

- When process P2 receives the P1 release resource message, it removes **1:P1** request message from its queue.

2:P2	3:P1

Fig 2(d): Request queue for process P2 after removing 1:P1

- Since process P1 has 1:P1 as the head of the queue and P2 has 2:P2 as the head of the queue, both the process enter critical section, thus violating safety.

Problem 3: Testing correctness and comparing performance

- **Testing Correctness:**
    - A test process (test.da) is created that receives messages from all other processes, record information regarding critical sections and resource releases.
    - The test process detects liveness and safety violations.
    - Each process sends 2 types of messages to test process:
        - Critical section message (CS) on entering critical section
        - Release resource message (RELEASE)
    - Each process has a timeout of 3 seconds for critical section
    - If a process waits for more than 2 seconds before entering critical section, the test process flags it as a deadlock i.e. liveness violation.
    - The test program validates process logs and checks if there are two or more processes entering critical section without releasing the resource. It then flags it as safety violation
- **Performance comparison:**
    - main.da does several different runs varying a particular parameter, with multiple repetitions of each run, and measures total user time, total system time and total process time and total memory, and report statistics about them.
    - It uses controller.da to get performance statistics of different runs.
- **Requirements:**
    - Environment: **macOS** Darwin Kernel Version 16.7.0
    - It uses PrettyTable to create and display tables:

pip install PTable

- **Execute program**:

python -m da main.da p r n d a

- **Correctness example:**

Correctness:

Algorithm	Total Processes	Requests	Safety violation	Liveness violation
orig.da	2	2	False	False
spec.da	2	2	False	False
lamport.da	2	2	False	True
orig.da	1	4	False	False
spec.da	1	4	False	False
lamport.da	1	4	False	False
orig.da	2	3	False	False
spec.da	2	3	False	False
lamport.da	2	3	False	True
orig.da	3	3	False	False
spec.da	3	3	False	False
lamport.da	3	3	False	True
orig.da	3	4	False	False
spec.da	3	4	False	False
lamport.da	3	4	True	True
orig.da	3	3	False	False
spec.da	3	3	False	False
lamport.da	3	3	False	True

- **Performance comparison:**

Performance comparison: Varying request numbers

Algorithm	Total Processes	Requests	Total user time	Total system time	Total process time	Total memory
orig.da	3	2	0.02935	0.00593	0.03528	20037632.0
spec.da	3	2	0.02844	0.00472	0.03316	20142080.0
lamport.da	3	2	0.03012	0.00566	0.03578	20088832.0
orig.da	3	4	0.0552	0.00752	0.06272	20011008.0
spec.da	3	4	0.0628	0.00668	0.06948	20133888.0
lamport.da	3	4	0.0554	0.0076	0.063	20217856.0

Performance comparison: Varying process numbers

Algorithm	Total Processes	Requests	Total user time	Total system time	Total process time	Total memory
orig.da	3	4	0.04877	0.00649	0.05526	20041728.0
spec.da	3	4	0.06091	0.00648	0.06739	20164608.0
lamport.da	3	4	0.06186	0.00801	0.06987	20166656.0

References:

- https://www.microsoft.com/en-us/research/uploads/prod/2016/12/Time-Clocks-and-the-Ordering-of-Events-in-a-Distributed-System.pdf
- https://dl.acm.org/citation.cfm?id=3231109
- https://sites.google.com/site/distalgo/
- cse535 mailing group
- Discussed with Jatin Garg, Abhinav Adarsh and Siddesh Shinde