SHIVANSHI GUPTA
SBU ID - 112079392
ASSIGNMENT - 3

Question 1. Summarize and compare the specifications of the algorithms and of the correctness properties. Include at least 3 properties: specification sizes; ease of understanding; how closely are different aspects of algorithms followed; and what properties are specified and in what way?

SPECIFICATION SIZES:

Algorithm 1: 88
Algorithm 2: 70
Algorithm 3: 89

EASE OF UNDERSTANDING:

***Algorithm 1*:** The spec demonstrates the different parts of the algorithm efficiently. The Init functionality starts with keeping all processes and clocks to 0. The Request starts with every process checking for its value of request, which is bounded to be 0. If it is 0, then it sends the request message with its clock-time to other processes and makes req[p][p] = clock[p].
On receiving the request from process q, the process p makes req[p][q]=c. The clock is set for process p, based on the clock value of the message received. In network, the process p appends the acknowledgement message and sends it to process q. The process ‘p’ on receiving acknowledgement from ‘q’ adds {q} to its acknowledgement and removes the message type of ‘ack’ from the network[q][p].
The process ‘p’ enters the critical section after checking if he beats the process ’q’ in clock value.
It then exits the Critical Section, broadcasting the messages ‘release’ to all processes. Also, it makes request as 0 and makes acknowledgement set as empty.
On receiving the release message from the process ’q’, the process ’p’ makes the request type message in network[q][p] as 0, removes it from the head and keep the clock, critical section, ack unchanged.

***Algorithm* 2**: Initializing the state of every process as “idle”. When a process requests its state is “waiting”. Clock is set to any value ‘n’. The message with command “acquire” is sent to all process in Proc except ‘p’. In requestSet, the message is appended for process ‘p’. LastTSent for ‘p’ becomes the timestamp at which the message was sent i.e ‘n’.
When the process receives a request message from another process, sets clock[p] as msg.TS if msg.TS>clock[p] else it remains clock[p]. It sends the acknowledgement message by appending it in msgQTail. LastTReceived is set to the msg.TS.
When the process “acquires” the critical section, it changes its state from “waiting” to “owner” if: pReq < timestamp of the last received message.
When the process “releases” the critical section, The state is changed from “owner” to “idle” and the message with command “release” is sent to all processes. LastTSent is set to clock[p] for all process q in proc except p.

***Algorithm 3*:** The spec demonstrates the mutex algorithm as a process with two categories: communicator and sites.

1. The process starts with making its status as “start”. It then goes in “ncrit” and “try” where it sends the message of kind - “request” to all the processes with its timeclock.
2. The process then enters the critical section by checking the following conditions:

2.1 Length of request = 0

2.2 Head of requestQ is the process itself.

2.3 The process has obtained the acknowledgement from all the other sites.

3. The process then makes its status as = “crit” if all the above criterias satisfy. It completes its work and turns the status = “exit”
4. For exiting, the process, broadcasts the message – “free” to all the other processes and changes itself to “start” again.

HOW CLOSELY ARE DIFFERENT ASPECTS OF THE ALGORITHM FOLLOWED:

Algorithm 1: 1st point of Lamport Mutex Followed: Send request message to all processes: The aspect is followed by broadcasting the request message to all processes, keeping the clock[p] as the timestamp.

2nd point of Lamport Mutex Followed: Sending Acknowledgement: The aspect Is followed by the receiving the request message and sending acknowledgement to the process which has requested.

3rd point of Lamport Mutex. Followed: Sending Release message: The aspect is followed by the process by removing the request from its request set and broadcasting the release message.

4th point of Lamport Mutex Followed: Receiving Release message: The aspect is followed by the process by removing the request message from its network. This time it removes “**ANY 1**” message of the requesting process.

5th point of Lamport Mutex Followed: Checking the conditions: The process checks if “beats (p,q)” is being followed.

Algorithm 2: 1st point of Lamport Mutex followed: The request is being sent to other processes except P and reqset[p].

2nd point of Lamport Mutex Followed: Acknowledgement is being sent to the requesting process after the message is appended in its own msgQ’.

3rd point of Lamport Mutex Followed: P’s request is removed from reqSet[p]. This is done in acquire itself. Message “release” sent to all processes except p.

4th point of Lamport Mutex followed: for “Release”, **any all** requests of releasing process is removed from reqSet[p].

5th point of Lamport Mutex followed: pReq is checked if it is the earliest request in the reqSet[p]. Also, it is checked that p must have received a message at a later time stamp than pREq from every other process.

Algorithm 3: 1st point of Lamport Mutex followed: Send request message to all processes: It broadcasts the message with type = "request" with clock[self] as the timestamp.

2nd point of Lamport Mutex followed: Acknowledgement Sending: If the message received by the process has "request", the acknowledgement is sent on the network

3rd point of Lamport Mutex followed: The release of the process is followed by sending "free" message to all the processes

4th point of Lamport Mutex Followed: After getting the "free" message, the process removes the request from its queue.

5th point of Lamport Mutex Followed: checking whether the head in ReqQ is self or not makes is what the spec follows.

WHAT PROPERTIES ARE FOLLOWED AND IN WHAT WAY:

SAFETY:

***Algorithm* 1**: The spec follows mutex or safety by specifying, that if a process 'p' and 'q' are in critical section, then both p and q are same. This means that any particular time, the critical section cannot be held by more than one process.

Mutex == \A p,q in crit : p=q

Algorithm2: The spec follows the mutex property by specifying that if process p and q are in a critical section at same time then, "p" is bound to be "q". That is p and q are the same processes and hence, there is just one process which is in critical section at any given time.

Mutex == \A p,q \in Proc:

(state[p] = "owner"/\ state[q] = "owner") => p=q

Algorithm3: The spec follows mutex or safety by specifying if two processes have ps[s] = crit and ps[t] = crit, it has to be s=t which implies that only 1 process can be in critical section at a particular time.

Mutex == \A s,t \in Sites: pc[s] = "crit" /\ pc[t] ="crit" => s=t

LIVENESS:

***Algorithm 1*:** The spec follows liveness by specifying, that if there is a process in PROC, it should EVENTUALLY GO IN THE CRITICAL SECTION. Therefore, maintaining the system liveness.

Liveness == \A p \in Proc: <>(p \in crit)

***Algorithm 2*:** The spec follows the liveness condition by specifying that if there is a process in the state "waiting", then eventually it will become the "owner" i.e. eventually it will go in critical section.

AcquiringEventually == \A p \in Proc: [](state[p] = "waiting") => <>(state[p] = "owner")

Algorithm 3*:* The spec follows liveness by specifying that if there is a process which has the status as “enter” should lead to the process in “critical section”.

Liveness == \A s \in Sites: pc[s] = “enter” ~>pc[s] = “crit

==========
==========
==========

Solution 2:

SYSTEM SPECIFICATIONS - Windows 10 10.0 amd64, Oracle Corporation 1.8.0_181 x86_64

ALGORITHM 1 - Lamport Mutex

No. of processes	Max Clock	Time taken to execute	Deadlock Achieved	Safety Violated	Liveness Violated	Number of States searched	Off Heap Memory	Ease of Setting (parameters)	Heap memory
2	5	12306 ms	No deadlock	Not violated	Yes Violated	1326	1342 MB	2	597 MB
2	10	9482ms	No deadlock	Not Violated	Yes violated	6061	1343MB	2	597 MB
2	7	6981ms	No deadlock	Not Violated	Yes Violated	2182	1343MB	2	597 MB
2	6	8582ms	No Deadlock	Not Violated	Yes Violated	2001	1343MB	2	597 MB
2	9	7051ms	No Deadlock	Not Violated	Yes Violated	4842	1343MB	2	597 MB
3	2	6799 ms	No Deadlock	Not Violated	Yes Violated	2203	1343MB	2	597 MB
3	3	8116ms	No Deadlock	Not Violated	Yes Violated	41533	1343Mb	2	597 MB
3	4	9677ms	No Deadlock	Not Violated	Yes Violated	276114	1343Mb	2	597 Mb

4	2	7610ms	No Deadlock	Not Violated	Yes Violated	68901	1343MB	2	597 MB

ALGORITHM 2 - TIME CLOCKS

Proc	(p,q)	Time taken to execute	Deadlock Achieved	Safety Violated	Liveness Violated	Number of States searched	Off Heap Memory	Ease of Setting (parameters)	Heap memory
{1,2} n={1..10}	p<q	1978 ms	Yes deadlock	Not violated	Not Violated	2978296	1342 MB	2	597 MB
{1,2,3} n={1..10}	p<q	106940 ms	Yes Violated	Not Violated	Not Violated	19377198	1343MB	2	597 MB
{1,2} n={1..3}	p<q	21625 ms	Yes Violated	Not Violated	Not Violated	75382	1343MB	2	597 MB

ALGORITHM 3 - MUTEX LAMPORT

No. of processes	Max Clock	Time taken to execute	Deadlock Achieved	Safety Violated	Liveness Violated	Number of States searched	Off Heap Memory	Ease of Setting (parameters)	Heap memory
2	5	12634 ms	No deadlock	Not violated	Yes Violated	23888	1343MB	2	597 MB
2	10	6614 ms	No deadlock	Not Violated	Yes violated	71395	1343MB	2	597 MB
3	5	1013548 ms	No deadlock	Not Violated	Yes Violated	48063817	1343MB	2	597 MB
3	7	8007095 ms	No Deadlock	Not Violated	Yes Violated	531442096	1343MB	2	597 MB

EXTRA CREDIT: You can run TLC or other tools on other good specifications of distributed mutual exclusion algorithms written in TLA+ or other formal specification languages, and compare with running TLC on the 3 specifications given.

ALGORITHM -4 : We took this algorithm from the internet :
Reference :

https://github.com/weituo12321/Asynchronous_Systems/tree/master/Week3%20Distributed%20mutex%20specifications

RICARTA AGGRAWALA TOKEN ALGORITHM:

On running the algorithm this was our recordings:

No. of processes	Max Clock	Time taken to execute	Mutex Violated	Number of States searched	Off Heap Memory	Ease of Setting (parameters)	Heap memory
2	2	15705 ms	violated	22517	1343MB	3	597 MB
2	3	3997ms	Violated	22910	1343MB	3	597 MB
2	9	4779ms	violated	22801	1343MB	3	597 MB
3	10	78242ms	violated	906911	1343MB	3	597 MB
3	5	74438ms	violated	938263	1343MB	3	597MB